Computer Corporation of America

575 Technology Square
Cambridge
Massachusetts 02139 617-491-3670

LEVEL II

(12)

AD A064989

DDC FILE COPY

(6) Program Listings for SWF-D: The Signal Waveform File Demon.

(10) Joanne Z. Sattley

(9) Technical Report, (14) CCA-79-10

(11) 31 January 1979

(12) 133 p.

DDC RECEIVED FEB 28 1979 A

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

(15) This research was supported by the Advanced Research Projects Agency of the Department of Defense under ~~Contract No.~~ N0014-78-C-0246, ARPA Order No. - 3540. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

387 285

LB

79 02 07 032

Program Listings for SWF-D: The Signal Waveform File Demon

Joanne Z. Sattley

Technical Report CCA-79-10

January 31, 1979

This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. N00039-78-C-0246, ARPA Order No. 3540. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

Acknowledgement

Special thanks are due to Donald E. Eastlake of Computer Corporation of America and to Leslie J. Turek of Lincoln Laboratory, Massachusetts Institute of Technology, both for their contributions to the concepts which have been implemented and for originating much of the Datalanguage used for communicating with the Datacomputer by the SWF-D program.

Table of Contents

1. Introduction

This document contains listings of the SWF-D program source-code and data definitions; it is intended to serve as a companion document to the Report on the Implementation and Test of SWF-D: The Signal Waveform File Demon, which is distributed as CCA Technical Report #CCA-79-09.

The source-code is divided here into sections, each of which represents a separately compiled program module and, to facilitate referencing, subsection numbers have been assigned to several of the more interesting routines within the modules. Global data definitions, constants and variables are listed in the Appendix.

Two pre-existing CCA-developed programs which were created as parts of other projects are not included here. These are:

. DCSTAT, the Datacomputer status checking program, and

. BDSUBR, a package of utility routines for interfacing to the Datacomputer.

DCSTAT is a free-standing program which is loaded dynamically by the SWF-D program; the status information is transmitted between the two programs via a Tenex file. The BDSUBR package, on the other hand, is loaded into core as an integral part of the SWF-D program, permitting Datacomputer communications to be handled as ordinary subroutine calls. Listings are readily available from CCA upon request.

The SWF-D program is written in the BCPL programming language. The resultant code, though quite readable for the most part, requires considerable knowledge of the language's unusual treatment of data types for complete comprehension -- and, thus, cannot be recommended to the casual reader. The program maintainer, however, should find the contents rather useful even though the most up-to-date versions are to be found online.

Generating an executable SWF-D program involves compiling each module, loading together all the REL files using the Tenex LINK10 loader, and then saving the resultant core image. The operating characteristics are covered in the implementation and test report.

2. The Main Control Module

```
// SWF-D Program: Main Module

get     "<CCA-SWF>SWFHEAD.BCP"

external        {       CheckDC         }
external        {       CheckL          }
external        {       EVENTS          }
external        {       GAvail          }
external        {       MARK            }
external        {       MOVES           }
external        {       OKGoQ           }
external        {       RportL          }
external        {       UPDAT           }

static  {stat

 dcstatBCPL     :       vec     512
 dcstatBUFF     :       vec     1000
 dcstatJFN      :       nil
 dcstatPTR      :       nil
 RDateSTR       :       vec     15
 smJFN          :       nil
 smFRKH         :       nil
 jsACs          :       vec     10
 MsgsBuffer     :       vec     512
 loadlbl        :       nil
 loadlvl        :       nil
 datelbl        :       nil
```

```
 datelvl        :       nil
 wrkvec         :       vec     10

}stat

let Start() be

{st

JSYS(jsRESET)
jsACs{1 := #636746,,#154400             // program ID
JSYS(jsSETNM,jsACs)

RportL("NEW SESSION")

// Create fork & load with DC status checker.  Problems encountered
// here should be checked out and the error condition corrected prior
// to restarting SWF-D.

jsACs{1 := #200000,,0           // capabilities = thisfork
if JSYS(jsCFORK,jsACs) eq failed  then  {forkerr
    RportL("CFORK failure; restart SWF-D")
    finish
}forkerr
smFRKH := jsACs{1
jsACs{1 := #100001,,0           // old file
jsACs{2 := POINT(7, '<SUBSYS>DCSTAT.SAV')
if  JSYS(jsGTJFN, jsACs) eq failed  then  {jfnerr
    RportL("GTJFN failure; restart SWF-D")
    finish
}jfnerr
smJFN := jsACs{1
jsACs{1 := smFRKH,,smJFN
JSYS(jsGET, jsACs)
```

```
// set up interrupt programs

let cntlLchan := FreeTICh()              // set up int for ^L
ATI($^L,cntlLchan)
PSISetCh(1,cntlLchan,LoadWorkSchedule)
let cntlSchan := FreeTICh()              // set up int for ^S
ATI($^S,cntlSchan)
PSISetCh(1,cntlSchan,SetStationData)
PSIOn()

// set wake-up control bits

jsACs¦1 := INPUT
JSYS(jsRFMOD,jsACs)
(jsACs¦2)<<TT.WAK := #77
JSYS(jsSFMOD,jsACs)

loadlbl := SWFTop
loadlvl := Level()
datelbl := SWFTop
datelvl := Level()



// Check for existence of SWF-D.WORK%SCHEDULE file;
// This is a clue which indicates whether the program is
// starting (and needs initializing) or restarting.

// jsACs¦1 := ofOldFile
// jsACs¦2 := POINT(7,'SWF-D.WORK%SCHEDULE')

// if JSYS(jsGTJFN,jsACs) ne failed then ¦restart
//    TaskJFN :=  rh jsACs¦1
//    goto SWFTop                        ¦restart
```

```
//TaskJFN := CreateOutput("SWF-D.WORK%SCHEDULE",7)

// initialize working parameters

SetStationData()
LoadWorkSchedule()
dcstatJFN := 0
DCicp := false
MARK(TaskStatus,InitCompleted)
RportL("Initialization complete")
Logpg>>Log.NextTask := TopoftheQueue

// Wait 1 hour before performing any tasks.  This serves two
// purposes:  (1) it allows the Datacomputer to restart itself
// and to perform its directory cross-checking more rapidly, and
// (2) it provides for resetting the program work schedule, the
// station data, and/or other task-dependent data by means of the
// interrupt processing routines.

LIMBEAUX(1)
RportL("Beginning task processing")

SWFTop:
OKGoQ()                          // check working conditions
let nt := Logpg>>Log.NextTask    // pick up first|next task in queue
switchon nt into {ntask

    case TopoftheQueue:
    case Restart:       Logpg>>Log.Taskix := 0
                        RportL("Top of the task queue")
                        endcase

    case Limbo:         Logpg>>Log.Taskix := Logpg>>Log.Taskix + 1
                        LIMBEAUX(Logpg>>Log.Task^(Logpg>>Log.Taskix))
                        endcase
```

```
    case GetArrivals:   if ~ EVENTS() then  {gaf
                            LIMBEAUX(2)
                            goto SWFTop }gaf
                        endcase

    case AppendSWF:     if ~ MOVES() then {mof
                            LIMBEAUX(2)
                            goto SWFTop }mof
                        endcase

    case UpdateESF:     if ~ UPDAT() then  {upf
                            LIMBEAUX(2)
                            goto SWFTop }upf

                        endcase

    case GenSegAvailMap:        if ~ GAvail() then {gaf
                                    LIMBEAUX(2)         // wait 2 hrs
                                    goto SWFTop   }gaf    // & try again

    default:                    endcase

}ntask

// When task has completed a self-appointed quota, Taskix is
// bumped & the next time through the loop, another task will
// be selected.

Logpg>>Log.Taskix := Logpg>>Log.Taskix + 1
if Logpg>>Log.Taskix > Logpg>>Log.Tasklim then Logpg>>Log.Taskix := 1
Logpg>>Log.NextTask := Logpg>>Log.Task^(Logpg>>Log.Taskix)
RportL("Next task:")
goto SWFTop

}st
```

2.1 LoadWorkSchedule, Control-L Interrupt Processor

```
// LoadWorkSchedule is the control-L interrupt processor.
// It maintains the SWF-D.WORK%SCHEDULE Tenex file and creates one
// as part of the program initialization sequence if one does not exist.

and let LoadWorkSchedule(l,v,lvpc) be

{lws

RportL("Loading work schedule")
// set wake-up control bits

jsACs¦1 := INPUT
JSYS(jsRFMOD,jsACs)
(jsACs¦2)<<TT.WAK := #77
JSYS(jsSFMOD,jsACs)

let WorkJFN := nil

let Tch,tix,numb := nil,nil,nil

jsACs¦1 := ofOldFile\ofAssignOnly
jsACs¦2 := POINT(7,'SWF-D.WORK%SCHEDULE')

if JSYS(jsGTJFN,jsACs) ne failed then {is

    WorkJFN := rh jsACs¦1
    jsACs¦2 := #440000,,#303000
    if JSYS(jsOPENF,jsACs) eq failed then {
        RportL("Check SWF-D.WORK%SCHEDULE file and restart program")
```

```
        finish }

SIN(WorkJFN,POINT(36,Logpg),512)

// Check for updates if processing interrupt - else
// return to calling program.

if numbargs < 3 then {    CLOSF(WorkJFN) ;    return  }

    jsACs¦1,jsACs¦2 := WorkJFN,0
    JSYS(jsSFPTR,jsACs)
    goto QueryL

}is

WorkJFN := CreateOutput("SWF-D.WORK%SCHEDULE",36)

LWSInit:

WriteS("*nReady to initialize work schedule*n")

// Initialize default values but save old ESFCurrentDate

{lwsinit

let tvec := vec 5
CopyString(lv (Logpg>>Log.ESFCurrentDate),tvec)

for ix := 1 to 512 do Logpg¦ix := 0

Logpg>>Log.Taskix := 0
Logpg>>Log.Tasklim := 0
Logpg>>Log.LoadLimit := 3.0
Logpg>>Log.NextTask := TopoftheQueue
```

```
// Reset ESFCurrentDate only if virgin Logpg

CopyString(tvec,lv (Logpg>>Log.ESFCurrentDate))
if Logpg>>Log.ESFCurrentDate eq 0 then

    {   Logpg>>Log.ESFCurrentDate := 1978,,#1001    }

Logpg>>Log.Interval := 1

}lwsinit

QueTop:

WriteS("Task := ")
Tch := PBIN()
switchon Tch into {tlp

 case $?:       {qmk

                Writech($*n)
                WriteS("*t?  =>  Display list of tasks*n")
                WriteS("*tE  =>  Scan ESF for arrivals*n")
                WriteS("*tG  =>  Generate segment availability map*n")
                WriteS("*tM  =>  Move waveforms*n")
                WriteS("*tU  =>  Update ESF*n")
                WriteS("*tW  =>  Wait (program delay) n hours*n")
                WriteS("*tT  =>  Go to top of task queue*n")
                WriteS("*tR  =>  Restart at top of task queue*n")
                WriteS("*n*tV  =>  View current task queue*n")
                WriteS("*tC  =>  Clear current task queue*n")
                WriteS("*tQ  =>  Quit {review SWF-D control variables}*n")

                }qmk

                endcase
```

```
case $r:
case $R:
case $t:
case $T:        WriteS("*tStart at top of queue*t[OK]*n")
                tix := Logpg>>Log.Tasklim + 1
                Logpg>>Log.Task^tix := Restart
                Logpg>>Log.Tasklim := tix
                endcase

case $e:
case $E:        WriteS("SF scan for arrivals*t[OK]*n")
                tix := Logpg>>Log.Tasklim + 1
                Logpg>>Log.Task^tix := GetArrivals
                Logpg>>Log.Tasklim := tix
                endcase

case $g:
case $G:        WriteS("enerate segment map*t[OK]*n")
                tix := Logpg>>Log.Tasklim + 1
                Logpg>>Log.Task^tix := GenSegAvailMap
                Logpg>>Log.Tasklim := tix
                endcase


case $m:
case $M:        WriteS("ove waveforms*t[OK]*n")
                tix := Logpg>>Log.Tasklim + 1
                Logpg>>Log.Task^tix := AppendSWF
                Logpg>>Log.Tasklim := tix
                endcase

case $w:
case $W:        WriteS("ait n hours*t[OK]*n")
                tix := Logpg>>Log.Tasklim + 1
                Logpg>>Log.Task^tix := Limbo
                tix := tix + 1
```

```
                WriteS("# hours = ")
                { let numb := ReadN(INPUT)
                Logpg>>Log.Task~tix := numb  }
                Logpg>>Log.Tasklim := tix
                endcase

  case $u:
  case $U:      WriteS("pdate ESF*t[OK]*n")
                tix := Logpg>>Log.Tasklim + 1
                Logpg>>Log.Task~tix := UpdateESF
                Logpg>>Log.Tasklim := tix
                endcase

ViewTQ:
  case $v:
  case $V:      WriteS("iew task queue*t[OK]*n*nTask-index*tTask*n*n")
                {plp
                let nt := nil
                for tix := 1 to Logpg>>Log.Tasklim do {dlp
                Writech($*t) ; WriteN(tix) ; Writech($*t)
                {ntlp  nt := Logpg>>Log.Task~tix + #60
                switchon nt into {

                 case $1: WriteS("Limbo*t")
                          tix := tix + 1          // to get n hours
                          WriteN(Logpg>>Log.Task~tix)
                          WriteS(" hours*n")
                          endcase

                 case $2: WriteS("Get Arrivals*n")
                          endcase

                 case $3: WriteS("Append to SWF*n")
                          endcase
```

```
                    case $4: WriteS("Update ESF*n")
                             endcase

                    case $6:
                    case $5: WriteS("Transfer to top of task queue*n")
                             endcase

                    case $7: WriteS("Generate SPDET Map*n")
                    default: endcase

                  }  }ntlp
                  }dlp
                  Writech($*n)
                  WriteS("Current task index = ")
                  WriteN(Logpg>>Log.Taskix) ; Writech($*n)
                  WriteS("Current task limit = ")
                  WriteN(Logpg>>Log.Tasklim) ; Writech($*n)   }plp
                  endcase

  case $c:
  case $C:        WriteS("lear task chain*n")
                  goto LWSInit
                  endcase

  case $q:
  case $Q:        WriteS("uit*t|OK - on to review control variables|*n")
                  goto QueryL
  default:
                  endcase

}tlp

goto QueTop

QueryL:
```

```
WriteS("Select item to print¦update:  ")
Tch := PBIN()
switchon Tch into ¦pulp

  case $?:      Writech($*n) .
                WriteS("*t?  =>  Display items*n")
                WriteS("*tI  =>  Set Interval for automatic program delay*n")
                WriteS("*tL  =>  Set load limit*n")
                WriteS("*tC  =>  Clear task chain*n")
                WriteS("*tN  =>  Next task check*n")
                WriteS("*tA  =>  Append to task queue*n")
                WriteS("*tE  =>  Set ESF Current Date*n")
                WriteS("*tW  =>  Set work schedule*n")
                WriteS("*tV  =>  View current task queue*n")
                WriteS("*tQ  =>  Quit ⌊return to task processing⌋*n")
                endcase

  case $i:
  case $I:      ¦ivl  WriteS("nterval = ")

SetINT:

                WriteN(Logpg>>Log.Interval)
                WriteS("*tChange it? ⌊Y¦N⌋  ")
                let ch := PBIN()
                if (ch eq $N \ ch eq $n) then ¦ WriteS("o*n") ; endcase ¦
                if (ch eq $Y \ ch eq $y) then ¦
                WriteS("es*n⌊OK⌋*tNew Interval = ")
                let numb := ReadN(INPUT)
                Logpg>>Log.Interval := numb  ¦
                WriteS("Interval reset to:  ")
                goto SetINT
                endcase
                ¦ivl
```

```
  case $l:
  case $L:      ¦ll
                WriteS("oad limit = ")
SetLL:
                jsACs¦1 := OUTPUT ; jsACs¦3 := 0
                jsACs¦2 := Logpg>>Log.LoadLimit
                JSYS(jsFLOUT,jsACs)
                jsACs¦1 := INPUT
                WriteS("*tChange it? ⌊Y¦N⌋*t")
                let ch := PBIN()
                if (ch eq $N \ ch eq $n) then ¦ WriteS("o*n") ; endcase }
                if (ch eq $Y \ ch eq $y) then ¦
                WriteS("es*n⌊OK⌋*tNew load limit:  ")
GetLL:
                if JSYS(jsFLIN,jsACs) eq failed then ¦
                WriteS("*nBad value - try again:*t")
                goto GetLL }
                Logpg>>Log.LoadLimit :=jsACs¦2
                WriteS("Load Limit reset to:  ")
                goto SetLL      }
                endcase
                }ll

  case $c:
  case $C:      WriteS("lear task chain*n")
                goto LWSInit
                endcase

  case $n:
  case $N:      ¦ntk  WriteS("ext task = ")

SetTSK:

                WriteN(Logpg>>Log.NextTask)
                WriteS("*n1 = Limbo, 2 = GetArrivals, 3 = AppendSWF, ")
                WriteS("4 = UpdateESF*n")
```

```
                WriteS("5 = TopoftheQueue, 6 = Restart, 7 = GenSegAvailMap*n")
                WriteS("*tChange it? [Y|N]  ")
                let ch := PBIN()
                if (ch eq $N \ ch eq $n) then { WriteS("o*n") ; endcase }
                if (ch eq $Y \ ch eq $y) then {
                WriteS("es*n[OK]*tNext task = ")
                let numb := ReadN(INPUT)
                Logpg>>Log.NextTask := numb  }
                WriteS("Next task reset to:  ")
                WriteN(Logpg>>Log.NextTask) ; Writech($*n)
                endcase
                }ntk

  case $a:
  case $A:      WriteS("ppend task*n")
                WriteS("Ready to append to task queue*n")
                goto QueTop
                endcase


 case $v:
 case $V:       goto ViewTQ
                endcase

ESFDate:
 case $e:
  case $E:      { WriteS("SFCurrentDate [day month year]  =  ")
                WriteN(Logpg>>Log.ESFCurrentDate.Day)
                Writech($-)
                WriteN(Logpg>>Log.ESFCurrentDate.Mo)
                Writech($-)
                WriteN(Logpg>>Log.ESFCurrentDate.Yr)
                WriteS("*nChange it? [Y|N]*t")
                let ch := PBIN()
                if (ch eq $N \ ch eq $n) then { WriteS("o*n") ; endcase }
                if (ch eq $Y \ ch eq $y) then {
```

```
                    WriteS("es*t⌊OK⌋*tNew date :=  ")
                    numb := ReadN(INPUT)
                    Logpg>>Log.ESFCurrentDate.Day := numb

                    numb := ReadN(INPUT)
                    Logpg>>Log.ESFCurrentDate.Mo := numb

                    numb := ReadN(INPUT)
                    Logpg>>Log.ESFCurrentDate.Yr := numb
                    Writech($E)
                    goto ESFDate
                  }
                  endcase
                  }

  case $w:
  case $W:        WriteS("ork schedule*n")
// interactively determines when SWF-D will¦will not work
// and reprograms the task queue accordingly. <<< unimplemented
                  endcase

  case $q:
  case $Q:        WriteS("uit ⌊resume task processing⌋*n")
                  goto LWSout
  default:
                  endcase

}pulp
goto QueryL
LWSout:
SOUT(WorkJFN,POINT(36,Logpg),512)
CLOSF(WorkJFN)
if numbargs < 3 then return
LongDebrk(lvpc,loadlbl,loadlvl)
}lws
```

2.2 SetStationData, Control-S Interrupt Processor

```
// SetStationData is the control-S interrupt processor but it can
// also be called as a simple subroutine to reload station data.  It
// maintains the SWF-D.STATION%DATA Tenex file and will create it as
// part of the program initialization sequence if one does not exist.
// This program provides for:
//
//      . acquiring information by station about SRO data stored on
//        the Datacomputer,
//
//      . printing current station data (i.e., station name, period
//        of the data stored for the station on the Datacomputer
//        (as advised by messages from ASL), and the date of the
//        last short-period detections file generated for the station.
//
//      . updating the station data per ASL advice,
//
//      . adding a new station, and
//
//      . deleting a station.
//

and let SetStationData(l,v,lvpc) be

{ssd

RportL("SWF-D acquiring Station data")

// set wake-up control bits
```

```
jsACs¦1 := INPUT
JSYS(jsRFMOD,jsACs)
(jsACs¦2)<<TT.WAK := #77
JSYS(jsSFMOD,jsACs)

// If the Tenex file SWF-D.STATION%DATA exists, SetStationData will
// initialize the StationData structure from the file.  Otherwise,
// it will interactively construct the dataset and create a new file.

let StationDataJFN := nil
let OPch := nil

jsACs¦1 := ofOldFile\ofAssignOnly
jsACs¦2 := POINT(7,'SWF-D.STATION%DATA')

if JSYS(jsGTJFN,jsACs) ne failed then {is
// Open file:

    StationDataJFN := rh jsACs¦1
    jsACs¦2 := #440000,,#303000
    if JSYS(jsOPENF,jsACs) eq failed then {
        RportL("Check SWF-D.STATION%DATA file and restart program")
        finish }

// Read in data and construct the StationData dataset.

SIN(StationDataJFN,POINT(36,Stations),512)

// Check for updates if processing interrupt - else
// return to calling program.

if numbargs < 3 then {    CLOSF(StationDataJFN) ;    return  }

    jsACs¦1,jsACs¦2 := StationDataJFN,0
    JSYS(jsSFPTR,jsACs)
    goto UpdateQ
```

```
}is

// Create new SWF-D.STATION%DATA file,

StationDataJFN := CreateOutput("SWF-D.STATION%DATA",36)

// then initialize dataset:

WriteS("*nReady to initialize data for Stations")

SetTop:
{inits

for i:= 1 to 512 do Stations¦i := 0
let ix := 1
let numb := nil
Stations>>StationData.AllStationsASLDate := 1978,,#1001

InitStations:

for i := 1 to 10 do wrkvec¦i := 0
WriteS("*nStation name:  ")
ReadWord(wrkvec)
CopyString( wrkvec, lv (Stations>>StationData.Station^ix.SName^1) )

WriteS("*nFrom date [day month year]:  ")
numb := ReadN(INPUT)
Stations>>StationData.Station^ix.FromDate.Day := numb

numb := ReadN(INPUT)
Stations>>StationData.Station^ix.FromDate.Mo := numb

numb := ReadN(INPUT)
Stations>>StationData.Station^ix.FromDate.Yr := numb
```

```
WriteS("*nTo date [day month year]:  ")
numb := ReadN(INPUT)
Stations>>StationData.Station^ix.ToDate.Day := numb

numb := ReadN(INPUT)
Stations>>StationData.Station^ix.ToDate.Mo := numb

numb := ReadN(INPUT)
Stations>>StationData.Station^ix.ToDate.Yr := numb

Stations>>StationData.Stationix := ix
ix := ix + 1

WriteS("*nMore? [Y|N]*t")
let ch := PBIN()
if ( ch eq $Y \ ch eq $y ) then { WriteS("es*n") ; goto InitStations }
if ( ch eq $n \ ch eq $N ) then { WriteS("o*n") }

}inits

// STATION%DATA now in core.  Check caller options to update, print, etc.

UpdateQ:

WriteS("*nSelect operation (? for options):*n")

OPch := PBIN()
switchon OPch into {oplp

 case $?:       {qmark

                Writech($*n)
                WriteS("*t?   =>  Display menu of operations*n")
                WriteS("*tP   =>  Print current info*n")
                WriteS("*tC   =>  Change ASL date for all Stations*n")
                WriteS("*tS   =>  Set SPDET date for all Stations*n")
```

```
                WriteS("*tU  =>  Update ASL data by Station*n")
                WriteS("*tA  =>  Add a new Station*n")
                WriteS("*tD  =>  Delete a Station*n")
                WriteS("*tI  =>  Initialize data by Station*n")
                WriteS("*tQ  =>  Quit [return to task processing]*n*n")
                }qmark
                endcase

 case $c:
 case $C:       WriteS("hange ASL date for all Stations to [day month year]:  ")
                { let numb := ReadN(INPUT)
                  Stations>>StationData.AllStationsASLDate.Day := numb
                  numb := ReadN(INPUT)
                  Stations>>StationData.AllStationsASLDate.Mo := numb
                  numb := ReadN(INPUT)
                  Stations>>StationData.AllStationsASLDate.Yr := numb  }

                WriteS("*t[OK]*n")
                WriteS("*nAllStationsASLDate = ")
                WriteN(Stations>>StationData.AllStationsASLDate.Day)
                Writech($*s)
                WriteN(Stations>>StationData.AllStationsASLDate.Mo)
                Writech($*s)
                WriteN(Stations>>StationData.AllStationsASLDate.Yr)
                Writech($*n)
                endcase

 case $s:
 case $S:       WriteS("et SPDET date for all Stations to [day month year]:  ")
                { let numb := ReadN(INPUT)
                  Stations>>StationData.AllStationsSPDETDate.Day := numb
                  numb := ReadN(INPUT)
                  Stations>>StationData.AllStationsSPDETDate.Mo := numb
                  numb := ReadN(INPUT)
                  Stations>>StationData.AllStationsSPDETDate.Yr := numb  }
```

```
                WriteS("*t⌊OK⌋*n")
                WriteS("*nAllStationsSPDETDate = ")
                WriteN(Stations>>StationData.AllStationsSPDETDate.Day)
                Writech($*s)
                WriteN(Stations>>StationData.AllStationsSPDETDate.Mo)
                Writech($*s)
                WriteN(Stations>>StationData.AllStationsSPDETDate.Yr)
                Writech($*n)
                endcase

case $u:
case $U:        WriteS("pdate ASL data*n")
                {ulp
                for ix := 1 to Stations>>StationData.Stationix do {
                WriteS(lv (Stations>>StationData.Station^ix.SName^1))
                WriteS(":*told date = ")
                WriteN(Stations>>StationData.Station^ix.ToDate.Day)
                Writech($*s)
                WriteN(Stations>>StationData.Station^ix.ToDate.Mo)
                Writech($*s)
                WriteN(Stations>>StationData.Station^ix.ToDate.Yr)
                WriteS("*tnew date ⌊day month year⌋ = ")
                let numb := ReadN(INPUT)
                Stations>>StationData.Station^ix.ToDate.Day := numb
                numb := ReadN(INPUT)
                Stations>>StationData.Station^ix.ToDate.Mo := numb
                numb := ReadN(INPUT)
                Stations>>StationData.Station^ix.ToDate.Yr := numb }

                }ulp
                endcase

case $p:
case $P:        WriteS("rint*n")
                {plp
                WriteS("*s*s*s*sStation*tFrom:*t*tTo:*t*tSPDET-Date:*n*n")
```

```
                for ix := 1 to Stations>>StationData.Stationix do
                { WriteS("*s*s*s*s")
                  WriteS(lv (Stations>>StationData.Station^ix.SName^1))
                  Writech($*t)
                  WriteN(Stations>>StationData.Station^ix.FromDate.Day)
                  Writech($-)
                  WriteN(Stations>>StationData.Station^ix.FromDate.Mo)
                  Writech($-)
                  WriteN(Stations>>StationData.Station^ix.FromDate.Yr)
                  Writech($*t)
                  WriteN(Stations>>StationData.Station^ix.ToDate.Day)
                  Writech($-)
                  WriteN(Stations>>StationData.Station^ix.ToDate.Mo)
                  Writech($-)
                  WriteN(Stations>>StationData.Station^ix.ToDate.Yr)
                  Writech($*t)
                  WriteN(Stations>>StationData.Station^ix.SPDETDate.Day)
                  Writech($-)
                  WriteN(Stations>>StationData.Station^ix.SPDETDate.Mo)
                  Writech($-)
                  WriteN(Stations>>StationData.Station^ix.SPDETDate.Yr)
                  Writech($*n)
                }
PrintAllASLDate:
                  WriteS("*nAllStationsASLDate = ")
                  WriteN(Stations>>StationData.AllStationsASLDate.Day)
                  Writech($-)
                  WriteN(Stations>>StationData.AllStationsASLDate.Mo)
                  Writech($-)
                  WriteN(Stations>>StationData.AllStationsASLDate.Yr)
                  Writech($*n)

PrintAllSPDETDate:
                  WriteS("*nAllStationsSPDETDate = ")
                  WriteN(Stations>>StationData.AllStationsSPDETDate.Day)
                  Writech($-)
```

```
                WriteN(Stations>>StationData.AllStationsSPDETDate.Mo)
                Writech($-)
                WriteN(Stations>>StationData.AllStationsSPDETDate.Yr)
                Writech($*n)

              }plp
              Writech($*n)
              endcase

case $a:
case $A:      WriteS("dd new Station:  Name = ")
              { for i := 1 to 10 do wrkvec|i := 0
                ReadWord(wrkvec)
                let ix := Stations>>StationData.Stationix + 1
                CopyString(wrkvec,lv (Stations>>StationData.Station^ix.SName^1))
                WriteS("*nFrom date [day month year]:  ")
                let numb := nil
                numb := ReadN(INPUT)
                Stations>>StationData.Station^ix.FromDate.Day := numb

                numb := ReadN(INPUT)
                Stations>>StationData.Station^ix.FromDate.Mo := numb

                numb := ReadN(INPUT)
                Stations>>StationData.Station^ix.FromDate.Yr := numb

                WriteS("*nTo date [day month year]:  ")
                numb := ReadN(INPUT)
                Stations>>StationData.Station^ix.ToDate.Day := numb

                numb := ReadN(INPUT)
                Stations>>StationData.Station^ix.ToDate.Mo := numb

                numb := ReadN(INPUT)
                Stations>>StationData.Station^ix.ToDate.Yr := numb
```

```
                Stations>>StationData.Stationix := ix
              }
              endcase

 case $d:
 case $D:     WriteS("elete Station Name:  ")

              {dlp
              let si := nil
              for i := 1 to'10 do wrkvec{1 := 0
              ReadWord(wrkvec)

              {outl
              {lookl
              for ix := 1 to Stations>>StationData.Stationix do {

              if ( Eqstr ( wrkvec,
              ( lv (Stations>>StationData.Station^ix.SName^1)))) then {
                        si := ix ; goto DeleteStation }
              } }lookl
              WriteS("*nStation ") ; WriteS(wrkvec) ; WriteS(" not found*n")
              endcase }outl
DeleteStation:
              if si < Stations>>StationData.Stationix then {replp
              let rix := Stations>>StationData.Stationix
              CopyString(lv (Stations>>StationData.Station^rix.SName^1),
                lv (Stations>>StationData.Station^si.SName^1))
              Stations>>StationData.Station^si.ToDate.Day :=
                Stations>>StationData.Station^rix.ToDate.Day
              Stations>>StationData.Station^si.ToDate.Mo :=
                Stations>>StationData.Station^rix.ToDate.Mo
              Stations>>StationData.Station^si.ToDate.Yr :=
                Stations>>StationData.Station^rix.ToDate.Yr
              }replp
```

```
                Stations>>StationData.Stationix :=
                    Stations>>StationData.Stationix  - 1
                }dlp
                endcase

 case $i:
 case $I:       WriteS("nitialize all Stations*n")
                goto SetTop
                endcase

 case $q:
 case $Q:       WriteS("uit*n")
                goto WhichExit
                endcase

 case $*n:
 default:       goto UpdateQ
                endcase

}oplp

goto UpdateQ

WhichExit:

// Write out new or updated SWF-D.STATION%DATA file page.

SOUT(StationDataJFN,POINT(36,Stations),512)
CLOSF(StationDataJFN)

if numbargs < 3 then return
LongDebrk(lvpc,datelbl,datelvl)

}ssd
```

2.3 MARK, Record Task Progress

```
// MARK maintains the SWF-D task chain.  This information is used
// to start and restart the various tasks.

and let MARK(ent,svc) be

{mark

switchon ent into {mi

    case TaskStatus:    {  let ix := Logpg>>Log.Taskix + 1
//                      if ix le Tix then {
//                      Logpg>>Log.Task^ix := svc
//                      Logpg>>Log.Taskix := ix
                        return  // }
                        endcase         }

  case StationsStatus:
                        { let jfnx := nil
                        jsACs¦1 := ofOldFile\ofAssignOnly
                        jsACs¦2 := POINT(7,'SWF-D.STATION%DATA')

                        if JSYS(jsGTJFN,jsACs) eq failed then {fl

                        RportL("Cannot find SWF-D.STATION%DATA")
                        finish  }fl

                        jfnx := rh jsACs¦1
                        jsACs¦2 := #440000,,#303000
```

```
                    if JSYS (jsOPENF,jsACs) eq failed then {
                    RportL("Cannot open SWF-D.STATION%DATA file")
                    finish  }

                      Stations>>StationData.AllStationsSPDETDate := svc

                      for ix := 1 to Stations>>StationData.Stationix do {
                       Stations>>StationData.Station^ix.SPDETDate := svc }
                      SOUT(jfnx,POINT(36,Stations),512)
                    EndWrite(jfnx)
                    jsACs¦1 := jfnx
                    JSYS(jsRLJFN,jsACs)
                      endcase }

  case ESFStatus:
                    { let jfnx := nil
                    jsACs¦1 := ofOldFile\ofAssignOnly
                    jsACs¦2 := POINT(7,'SWF-D.WORK%SCHEDULE')

                    if JSYS(jsGTJFN,jsACs) eq failed then {fl

                    RportL("Cannot find SWF-D.WORK%SCHEDULE")
                    finish  }fl

                    jfnx := rh jsACs¦1
                    jsACs¦2 := #440000,,#303000

                    if JSYS (jsOPENF,jsACs) eq failed then {
                    RportL("Cannot open SWF-D.WORK%SCHEDULE file")
                    finish  }

                      Logpg>>Log.ESFCurrentDate := svc
                      SOUT(jfnx,POINT(36,Logpg),512)

                    EndWrite(jfnx)
                    jsACs¦1 := jfnx
```

```
                        JSYS(jsRLJFN,jsACs)
                          endcase }

    default:            endcase

}mi

}mark
```

2.4 LIMBEAUX, Wait n Hours

```
// LIMBEAUX

and let LIMBEAUX(hrs) be

{1x

RportL("Program in limbo")
let time := 24*60                  // hrs to mins
if numbargs > 0 then time := hrs * 60
jsACs¦1 := time * (1000*60)        // mins to millisecs
JSYS(jsDISMS,jsACs)
OKGoQ()

}1x
```

2.5 OKGoQ, Wait for Low System Load

```
// OKGoQ is called whenever it is feasible to wait voluntarily
// for a low Tenex load average.  The program will wait until
// the load average is less than LoadLimit, checking it at two-
// minute intervals.  The initial value of LoadLimit is 3.0; it
// may be reset interactively by typing control-L while the
// program is running.

and let OKGoQ() be

{okg

Dumdeedum:

if ~ CheckL() then {
        Wait(2*1000*60) ; goto Dumdeedum }  // 2 mins

}okg
```

2.6 CheckL, Check Tenex Load Average

```
// CheckL is called to check the 1-minute load average.  It returns
// true¦false to the caller according as the load average is below¦above
// the pre-set maximum.


and let CheckL() := valof

{checkl

jsACs¦1 := #637163,,#644164              // SYS,,TAT
JSYS(jsSYSGT,jsACs)
jsACs¦1 := #14,,rh jsACs¦2               // 1-minute load average
if JSYS(jsGETAB,jsACs) eq failed then {
        RportL ("GETAB failed on load average")
        finish }
if jsACs¦1 ge Logpg>>Log.LoadLimit then resultis false
    resultis true

}checkl
```

2.7 CheckDC, DC-203 Datacomputer Status Checker

```
// CheckDC is used to check the status of the DC-203 Datacomputer
// operating on CCA-Tenex.  It calls the DCSTAT program which has
// been loaded into a sub-fork during SWF-D program initialization;
// interprets the response; and returns true¦false to its caller
// according as the Datacomputer is available or not.

and let CheckDC() := valof

{ckdc

RportL("Checking Datacomputer status")

if dcstatJFN ne 0 then {xold

    jsACs¦1 := dcstatJFN
    JSYS(jsRLJFN,jsACs)

}xold

jsACs¦1 := ofOldFile\ofAssignOnly
jsACs¦2 := POINT(7,'DCSTAT.OUT')

if JSYS(jsGTJFN,jsACs) eq failed then {fl

    dcstatJFN := CreateOutput("DCSTAT.OUT",7)
    goto CallDCSTAT                 }fl

dcstatJFN := rh jsACs¦1
jsACs¦2 := #070000,,#303000
```

```
if JSYS (jsOPENF,jsACs) eq failed then {
    RportL("Cannot open DCSTAT.OUT file")
    finish  }

CallDCSTAT:

jsACs¦1 := smFRKH
JSYS(jsGPJFN,jsACs)
rh jsACs¦2 := dcstatJFN
JSYS(jsSPJFN, jsACs)            // set fork's primary JFNs
jsACs¦2 := 0
JSYS(jsSFRKV, jsACs)    // start fork using entry vector
JSYS(jsWFORK, jsACs)            // wait for it to finish

CLOSF(dcstatJFN)
jsACs¦1 := #636746,,#154400             // program ID
JSYS(jsSETNM,jsACs)

dcstatJFN := FindInput("DCSTAT.OUT",7)

ReadDCSTAT:

let statchr,slvc,elvc,whycode := nil,nil,nil,0
for i := 1 to 1000 do dcstatBUFF¦i := 0
dcstatPTR := POINT (7,dcstatBUFF)
SIN(dcstatJFN,dcstatPTR,5000,$*1)

// Check first line of status data for special error messages:
// If the first character is not "]" then CheckDC will proceed
// to check the specific advice (enclosed in parentheses);
// it may be a notice of Tenex preventive maintenance, or
// a message indicating that the system is HEAVILY or SEVERELY
// LOADED, or that some hardware is OFF-LINE.  If any of these
// conditions is true, CheckDC indicates to its caller that it
// would be better to wait for better operating conditions than
// to proceed, and records the reason on the operations log.
```

```
statchr := ILDB(lv dcstatPTR)

if statchr ne $] then  {msgck

CheckMSG:
    whycode := TenexLoad
    ASCIZToString(dcstatBUFF,dcstatBCPL)
    if findsubstr(dcstatBCPL,"HEAVILY",lv slvc,lv elvc,1) then goto FailOUT
    if findsubstr(dcstatBCPL,"SEVERELY",lv slvc,lv elvc,1) then goto FailOUT
    whycode := HardwareProblem
    if findsubstr(dcstatBCPL,"OFF-LINE",lv slvc,lv elvc,1) then goto FailOUT
    dcstatPTR := POINT (7,dcstatBUFF)
    SIN(dcstatJFN,dcstatPTR,5000,$*1)
    statchr := ILDB(lv dcstatPTR)       // check for
    if statchr ne $] then goto CheckMSG    // more error messages

}msgck


// Check external Datacomputer job status:  If the string "EXISTS"
// does not appear, CheckDC assumes that that status of the Data-
// computer is "DOWN" or that system work is in progress.

whycode := DCQuestionable
ASCIZToString(dcstatBUFF,dcstatBCPL)
if ~ findsubstr(dcstatBCPL,"EXISTS",lv slvc,lv elvc,1) then goto FailOUT
dcstatPTR := POINT (7,dcstatBUFF)
SIN(dcstatJFN,dcstatPTR,5000,$*1)

// Check for suspension of TBM operations:  If the character "%"
// appears, CheckDC assumes that TBM operations on one or more drives
// has been suspended.  The program is not capable of determining
// whether the particular drives needed by SWF-D are usable and that
// the right tapes are mounted.  After noting the condition, program
// operation continues.
```

```
whycode := TBMstatus
ASCIZToString(dcstatBUFF,dcstatBCPL)
if findsubstr(dcstatBCPL,"%",lv slvc,lv elvc,1) then {continuing

    RportL("TBM operations on some drives are suspended")

    dcstatPTR := POINT (7,dcstatBUFF)
    SIN(dcstatJFN,dcstatPTR,5000,$*1)
    ASCIZToString(dcstatBUFF,dcstatBCPL)

}continuing

// Check for Datacomputer-going-down message:  If the character "!"
// appears, CheckDC assumes that the Datacomputer will be halted for
// a length of time.  The program checks the scheduled down-time against
// the current time; if the difference is less than 1 hour, it will
// inhibit starting up a Datacomputer session.

whycode := NotEnoughTimeLeft
if findsubstr(dcstatBCPL,"!",lv slvc,lv elvc,1) then {cktime

    if ~ findsubstr(dcstatBCPL,"AT",lv slvc,lv elvc,1) then goto FailOUT

}cktime

// Check internal Datacomputer job state:  If the string "Normal
// Operation" does not appear, CheckDC will prevent initiating
// Datacomputer sessions.

whycode := AbnormalDCState
if ~ findsubstr(dcstatBCPL,"NORMAL",lv slvc,lv elvc,1) then goto FailOUT

// CheckDC does not scan or interpret the DCSTAT Operator Status Message.

// Check socket status information for LISTENING|NOT LISTENING.
// If the Datacomputer is NOT LISTENING, CheckDC returns immediately
```

```
// to its caller so that the caller can check again after a brief
// interval.

whycode := NotListening
until EofFlg do {notq

dcstatPTR := POINT (7,dcstatBUFF)
SIN(dcstatJFN,dcstatPTR,5000,$*1)
ASCIZToString(dcstatBUFF,dcstatBCPL)
if findsubstr(dcstatBCPL,"NOT",lv slvc,lv elvc,1) then goto FailOUT

}notq

// OK to connect to Datacomputer

RportL("OK to connect to Datacomputer")
CLOSF(dcstatJFN) ; resultis true

// Wait for better operating conditions

FailOUT:

RportL("Not OK to connect to Datacomputer:")
switchon whycode into {whynot

    case TenexLoad:      RportL("Tenex load is too high.")
                         endcase
    case HardwareProblem:  RportL("Some hardware is off-line.")
                         endcase
    case DCQuestionable:  RportL("Datacomputer is not up.")
                         endcase
    case TBMstatus:      RportL("TBM operations are suspended.")
                         endcase
    case NotEnoughTimeLeft:  RportL("Datacomputer is going down soon.")
                         endcase
    case AbnormalDCState:  RportL("DC job is not in NORMAL state.")
```

```
                        endcase
    case NotListening:  RportL("Datacomputer is in NOT LISTENING state.")
                        endcase

}whynot

    CLOSF(dcstatJFN) ; resultis false

}ckdc
```

2.8 RportL, Write Operations Log

```
// RportL maintains a reliable record of SWF-D operations on the
// SWF-D.OPERATIONS Tenex file.  The file format is: date/time-stamp
// followed by space followed by ASCII string followed by CRLF.
// SWF-D.OPERATIONS may be examined, listed, and deleted as often
// as desired.  New versions are created automatically.

and let RportL(istg) be

{rptl

let RportLJFN := nil
jsACs¦1 := ofOldFile\ofAssignOnly
jsACs¦2 := POINT(7,'SWF-D.OPERATIONS')

if JSYS(jsGTJFN,jsACs) eq failed then {crl

    RportLJFN := CreateOutput("SWF-D.OPERATIONS",7) ; goto MakeNote  }crl

RportLJFN := rh jsACs¦1
jsACs¦2 := #070000,,#121000

if JSYS (jsOPENF,jsACs) eq failed then Help ("RportL problems")

MakeNote:
// get current date/time

MakeDate(RDateSTR,0)            // current date
```

```
// output date/time space istg CRLF

WriteS(RportLJFN,RDateSTR) ; Writech(RportLJFN,$*s)
WriteS(RportLJFN,istg) ; WriteS(RportLJFN,"*c*l")

EndWrite(RportLJFN)
jsACs¦1 := RportLJFN
JSYS(jsRLJFN,jsACs)

}rptl
```

3. The PESF-Checking Module

```
// SWF-D Program:  EVENTS Module

// EVENTS is responsible for sifting PESF files for arrivals marked by SDAC

get     "<CCA-SWF>SWFHEAD.BCP"

external        {       CheckDC         }
external        {       CheckL          }
external        {       DCLook          }
external        {       EVENTS          }
external        {       MARK            }
external        {       OKGoQ           }
external        {       RportL          }

static  {stat

 findany        :       nil
 ndays          :       nil
 ESFfrag        :       vec     100
 ArrJFN         :       nil
 ArrPTR         :       nil
 ArrPORT        :       "REQ"
 Tick           :       nil
 ESFMint        :       nil
 ESFDint        :       nil
 ESFYint        :       nil
```

```
 ESFyear          :       "0000"
 ESFmonth         :       "00"
 ESFday           :       "00"
 LOGINstr         :       "Login SDAC.CCA.SWF;*n"
 DTLbuff          :       vec     1000
 BCPLstr          :       vec     1000
 MsgsBuffer       :       vec     512
 Arrival          :       vec     512
 BasicESFName     :       "%TOP.SDAC.VELANET.PESF."
 ESFname          :       vec     100
 DCESFname        :       vec     100

 jsACs            :       vec     10
 sumcount         :       nil
 morearrivals     :       nil

}stat


let EVENTS() := valof

{events

MARK(TaskStatus,InGetEvents)

let edp := lv Logpg>>Log.ESFCurrentDate
let adp := lv Stations>>StationData.AllStationsASLDate
Tick := 0

// Adjust for valid next Event Summary File date

jsACs¦2 := (edp>>Date.Yr),,((edp>>Date.Mo) - 1)
jsACs¦3 := ((edp>>Date.Day) - 1),,0
jsACs¦4 := 0,,5
```

```
if JSYS(jsIDCNV,jsACs) eq failed then {fixdate

edp>>Date.Day := 1                          // new month
edp>>Date.Mo := edp>>Date.Mo + 1

if edp>>Date.Mo > 12 then {hnewyr

edp>>Date.Yr := edp>>Date.Yr + 1                    // new year
edp>>Date.Mo := 1

}hnewyr

}fixdate

ESFYint := edp>>Date.Yr
ESFMint := edp>>Date.Mo
ESFDint := edp>>Date.Day

// Check for whether there is work to do.


let DeltaYear := adp>>Date.Yr - edp>>Date.Yr
let DeltaMonth := adp>>Date.Mo - edp>>Date.Mo
if DeltaYear > 0 then DeltaMonth := DeltaMonth + 12
if DeltaMonth ge 1 then {
    ndays := ((DaysPerMonth|(edp>>Date.Mo)) - edp>>Date.Day) + 1
    goto DoESF
}
ndays := adp>>Date.Day - edp>>Date.Day

if ndays > 0 then goto DoESF

NoWork:

MARK(TaskStatus,EndGetArrivals)
RportL("ESF scanning is up-to-date")
```

```
resultis true


DoESF:

RportL("Scanning for arrivals")

if ~ DCicp then {

    OKGoQ()
    if ~ CheckDC() then {

        RportL("Waiting to check for ESF arrivals")
        resultis false  }

    ScriptJFN := CreateOutput("SWF-D.SCRIPT",7)
    startdc(ScriptJFN)
    RportL("Beginning Datacomputer session")
    senddc(LOGINstr)
    senddc("OPEN REQ;*c*l")
    DCicp := true       }

inttotxt (ESFYint, ESFyear)     // integer to text conversions
inttotxt (ESFMint, ESFmonth)
inttotxt (ESFDint, ESFday)

let rstg := vec 100
append(rstg,"Starting day.mo.yr = ",rstg)
append(rstg,ESFday,rstg) ; addch($.,rstg)
append(rstg,ESFmonth,rstg)
append(addch($.,rstg),ESFyear,rstg) ; append(rstg," for ",rstg)
let nstg := vec 5
RportL(append(append(rstg,inttotxt(ndays,nstg),rstg)," days",rstg))

// Construct arrivals file name
```

```
for i := 0 to 100 do ESFname¦i := 0
for i := 0 to 100 do ESFfrag¦i := 0
addch($Y,ESFfrag)
append(ESFfrag,ESFyear,ESFfrag)
append(ESFfrag,".M",ESFfrag)
if ESFMint < 10 then { addch($0,ESFfrag)  }
append(ESFfrag,ESFmonth,ESFfrag)

append(append(ESFname,BasicESFName,ESFname),ESFfrag,ESFname)


// Ready now to loop through ESF day files.

for esi := ESFDint to (ESFDint + ndays - 1) by 1 do {esfl

// Construct specific day filename

append(ESFname,".D",DCESFname)
append(ESFfrag,"%D",TenexFile)
if ESFDint < 10 then { addch($0,DCESFname) ; addch($0,TenexFile)  }
append(DCESFname,ESFday,DCESFname)
append(TenexFile,ESFday,TenexFile)
ESFDint := ESFDint + 1
inttotxt(ESFDint, ESFday)

// Check that file exists

senddc("LIST ") ; senddc(DCESFname) ; senddc(";*c*l")

if ~ DCLook() then { morearrivals := false ;  break esfl }

// Open day file:  OPEN %TOP.SDAC.VELANET.PESF.Ynnnn.Mnn.Dnn,SYN=ESF;

senddc("OPEN ") ; senddc(DCESFname) ; senddc(", SYN = ESF;*c*l")
```

```
if ~ DCLook() then { morearrivals := false ;  break esfl }

// cannot proceed if there are file problems

// Send Datalanguage to scan for arrivals

scriptdc(0)              // inhibit scripting

findany := true
if ~ GetEvents() then findany := false

scriptdc(ScriptJFN)              // resume scripting
senddc("CLOSE ESF;*c*l")

// Quit voluntarily after processing of 10 ESF day files if load average
// is high or Datacomputer is busy.

Tick := Tick + 1

if Tick < 10 then loop esfl
Tick := 0
if ~ CheckL() then { morearrivals := true ; break esfl }
if ~ CheckDC() then { morearrivals := true ; break esfl }

}esfl


// End Datacomputer session

if ~ findany then { RportL("No flagged arrivals.") ; goto EndESF }
    RportL("Found some!")


EndESF:
```

```
senddc("CLOSE %OPEN;*c*l")
RportL("Ending Datacomputer session")
quitdc()

EndWrite(ScriptJFN)
DCicp := false
let edp := lv Logpg>>Log.ESFCurrentDate
edp>>Date.Day := edp>>Date.Day + 1
MARK(ESFStatus,Logpg>>Log.ESFCurrentDate)
resultis ~ morearrivals

}events
```

3.1 GetEvents, Retrieve Flagged Requests from Datacomputer

```
// GetEvents is called with the appropriate ESF open; it merely
// fields the data between the Datacomputer and a Tenex file.
// Tenex files are created as needed; each Tenex filename is keyed
// to the relevant Datacomputer filename by means of the filename
// -extension field.  For Datacomputer filename "Ynnnn.Mnn.Dnn",
// the corresponding Tenex filename is "ARRIVALS.Ynnnn%Mnn%Dnn".


and let GetEvents() := valof

{gete

RportL("Scanning for arrivals in:")
RportL(DCESFname)
let rstg := vec 100

sumcount := 0
ArrJFN := 0

opendc(ArrPORT,36)
if not senddc("Inhibit 100,5 ;*c*l") then resultis false

senddc("FOR ESF WITH ANY ARRIVALS WITH WAVEFORMAVAIL EQ *'T*' *c*l")
senddc("*tFOR REQ,ARRIVALS  WITH WAVEFORMAVAIL EQ *'T*' *c*l")
senddc("BEGIN   EINDEX=EINDEX ")
senddc("*tEVENTNUM=EVENTNUM   AINDEX=AINDEX    STA=STA*c*l")
senddc("*tCHANTYPE=CHANTYPE  RATE=RATE  CHANID=CHANID*c*l")
senddc("*tGAIN=GAIN  COMP=COMP  DATASEGSTART=DATASEGSTART*c*l")
senddc("*tPHASEARR=PHASEARR     PHASEID=PHASEID   AMP=AMP  END;*c*l")
```

```
{morearr

ArrPTR := POINT(36,Arrival)
let retcount := getfromdc(0,ArrPTR,512,$^z)
sumcount := sumcount + retcount
if retcount eq 0 then break morearr

// Work to do!
// write out arrivals
// Create Tenex file for arrivals


if ArrJFN eq 0 then {
append(rstg,"ARRIVALS.",rstg)
TenexFile := changesubstr(TenexFile,".","%")
append(rstg,TenexFile,rstg)
ArrJFN := CreateOutput(rstg,36)
}
SOUT(ArrJFN,ArrPTR,512,$^z)

}morearr repeatwhile dcgetstate


Logpg>>Log.ESFCurrentDate.Yr := ESFYint
Logpg>>Log.ESFCurrentDate.Mo := ESFMint
Logpg>>Log.ESFCurrentDate.Day := ESFDint - 1
MARK(ESFStatus,Logpg>>Log.ESFCurrentDate)

if sumcount eq 0 then resultis false
if ~ DCLook() then resultis false
CLOSF(ArrJFN)
    jsACs|1 := ArrJFN
    JSYS(jsRLJFN,jsACs)
RportL(append(rstg,"*screated",rstg))
resultis true
}gete
```

4. The Waveform-Copying Module

```
// SWF-D Program: MOVES Module

// MOVES is used to append waveforms to the SWF and to
// simultaneously update the ESF.  The Datalanguage requests
// are "driven" by pre-processed lists of waveform segments.

get     "<CCA-SWF>SWFHEAD.BCP"

external        {       CheckDC         }
external        {       CheckL          }
external        {       DCLook          }
external        {       MARK            }
external        {       MOVES           }
external        {       OKGoQ           }
external        {       PrReq           }
external        {       PrPutL          }
external        {       PrPutS          }
external        {       RportL          }

static  {stat

 Component      :       nil
 compcount      :       nil
 stindex        :       nil
 SPDETopen      :       nil
 BasicNLPFname  :       "%TOP.SDAC.VELANET.NLPF."
 BasicNSPFname  :       "%TOP.SDAC.VELANET.NSPF."
```

```
BasicSWFname    :       "%TOP.SDAC.VELANET.PSWF."
BasicPESFname   :       "%TOP.SDAC.VELANET.PESF."
BasicSPDETname  :       "%TOP.SDAC.VELANET.SPDET."
LOGINstr        :       "Login SDAC.CCA.SWF;*n"
EventsFiles     :       'ARRIVALS.**'               // ASCIZ ARRIVALS.*
LPArrivalsFiles:        'LP-ARRIVALS.**'            // ASCIZ LP-Arrivals.*
SPArrivalsFiles:        'SP-ARRIVALS.**'            // ASCIZ SP-Arrivals.*
EventsJFN       :       nil
LPArrivalsJFN   :       nil
SPArrivalsJFN   :       nil
SaveAJFN        :       nil
SPDetJFN        :       nil
SoutJFN         :       nil
LoutJFN         :       nil
SWF             :       vec     50
NXPF            :       vec     50
NSPF            :       vec     50
NLPF            :       vec     50
PESF            :       vec     50
SWFportL        :       "PUTL"
SWFportS        :       "PUTS"
jsACs           :       vec     10
CFname          :       vec     25
CFnameBCPL      :       vec     50
SPDETname       :       vec 20
NXPFyear        :       nil
NXPFmonth       :       nil
NXPFday         :       nil
A               :       vec     512             // arrivals input area
Rptr            :       nil
Pptr            :       nil
CSecInDay       :       nil
NumStg          :       vec 20
PrecedingCSeconds:      nil
FollowingCSeconds:      nil
nsta            :       nil
```

```
 LPBytesMoved    :       nil
 SPBytesMoved    :       nil

}stat


let MOVES() := valof

{moves

// Check for input ARRIVALS.* Tenex files.  If found, CRInput will create
// input files for moving available LP & SP data

if ~ CRInput() then { RportL("No new waveforms to move") }

// But old requests may not yet have been processed -
// Check for LP-ARRIVALS.* Tenex files;
// if none, there's no work to do on long-period files.

jsACs¦1 := #001101,,0           // #100101,,0    ?
jsACs¦2 := POINT(7,LPArrivalsFiles)

if JSYS (jsGTJFN,jsACs) eq failed then {
        RportL("No LP waveforms to move")
        goto CheckSP    }

{doinLP

MARK(TaskStatus,AppendingSWF)
RportL("Moving waveforms")



LPArrivalsJFN := rh jsACs¦1
```

```
if ~ DCicp then {

    OKGoQ()
    if ~ CheckDC() then {

        RportL("Waiting to move LP waveforms")
        resultis false  }

    ScriptJFN := CreateOutput("SWF-D.SCRIPT",7)
    startdc(ScriptJFN)
    RportL("Beginning Datacomputer session")
    senddc(LOGINstr)
    DCicp := true       }


jsACs¦1 := POINT (7,CFname)
jsACs¦2 := LPArrivalsJFN
jsACs¦3 := #000100,,0           // get extension field only
JSYS(jsJFNS,jsACs)

ASCIZToString(CFname,CFnameBCPL)
let rstg := vec 100
append(rstg,"Working from file: ",rstg)
RportL(append(append(rstg,"LP-ARRIVALS.",rstg),CFnameBCPL,rstg))

// construct variable portion of current file, a name of the form Ynnnn.Mnn.Dnn
// from the form Ynnnn%Mnn%Dnn

changesubstr(CFnameBCPL,"%",".")
append(SWF,BasicSWFname,SWF)
append(SWF,CFnameBCPL,SWF)

append(PESF,BasicPESFname,PESF)
append(PESF,CFnameBCPL,PESF)
```

```
if debugging then
    {
        SWF := "SWF"
        PESF := "ESF%UNINVERTED"
    }


MovesNLPF:


// Construct NLPF name


for i := 1 to 50 do NXPF¦i := 0
append(NXPF,BasicNLPFname,NXPF)
append(NXPF,CFnameBCPL,NXPF)

senddc("OPEN ") ; senddc(NXPF) ; senddc(" READ, SYN=NLPF;*c*l")
senddc("OPEN ") ; senddc(PESF) ; senddc(" WRITE, SYN=PESF;*c*l")
senddc("OPEN ") ; senddc(SWF) ; senddc(" APPEND, SYN=SWF;*c*l")

    senddc("OPEN PUTL;*n")
    opendc(SWFportL,36)

// Datalanguage is sent to update the PESF and to append to the
// PSWF simultaneously.  These are done together to ensure that the
// PESF and the PSWF files will remain in synch.

// The request as formulated assumes that the NLPF window is
// quantized to minutesand it will work for a day's worth of
// NLPF data if the waveform is not split across a day boundary.

// The Datalanguage request uses one PORT and three FILEs.
```

```
senddc("BEGIN   FOR X IN PUTL.ESFPUT*c*l")
senddc("*tUPDATE Y IN PESF WITH Y.EINDEX EQ X.EINDEX*c*l")
senddc("*tUPDATE Z IN Y.ARRIVALS WITH Z.AINDEX EQ X.AINDEX*c*l")
senddc("*tBEGIN         Z.DATASEGSTART=X.DATASEGSTART  Z.AMP=0*c*l")
senddc("*tZ.WAVEFORMAVAIL='Y'   END*c*l")
senddc("APPEND A IN SWF, B IN PUTL.SWFPUT*c*l")
senddc("BEGIN       A.EVENTID=B.EVENTID*c*l")
senddc("*tA.STA=B.STA     A.CHANTYPE=B.CHANTYPE*c*l")
senddc("*tA.RATE=B.RATE     A.CHANID=B.CHANID*c*l")
senddc("*tA.GAIN=B.GAIN     A.COMP=B.COMP*c*l")
senddc("*tA.START=B.START     A.DATAFORMAT=*'G*'*c*l")
senddc("*tA.SCALEFACTOR=B.SCALEFACTOR*c*l")
senddc("*tFOR C IN NLPF WITH C.STA EQ B.STA*c*l")
senddc("*tFOR D IN C.DATA WITH D.INDEX GE B.STARTI AND D.INDEX LE B.ENDI*c*l")
senddc("*tFOR E IN A.TIMESERIES, F IN D.TIMESERIES WITH B.TYP EQ F.TYPE*c*l")
senddc("*tE.DATUM=F.DATUM       END     END;*c*l")

jsACs¦1 := LPArrivalsJFN
jsACs¦2 := #440000,,#303000
if JSYS(jsOPENF,jsACs) eq failed then {
        RportL("Failure opening LP-Arrivals input file")
        resultis false  }


LPBytesMoved := 0

{lputl
for i := 1 to 512 do A¦i := 0
SIN(LPArrivalsJFN,POINT(36,A),24,$^z)
let BytesSent := puttodc(0,POINT (36,A),-24)

LPBytesMoved := LPBytesMoved + BytesSent

}lputl repeatwhile dcputstate & ~ EofFlg
```

```
endputdc()

senddc("CLOSE SWF;CLOSE PESF;CLOSE NLPF;*c*l")

// check LPBytesMoved

RportL("End LP moves")

// Close and delete LP-Arrivals file

CLOSF(LPArrivalsJFN)

}doinLP


// NSPF next

CheckSP:
// check for SP-ARRIVALS.* Tenex files
// if none, there's no work to do on short-period files

jsACs|1 := #001101,,0           // #100101,,0    ?
jsACs|2 := (POINT7x0,,SPArrivalsFiles)

if JSYS (jsGTJFN,jsACs) eq failed then {
        RportL("No SP waveforms to move")
        goto EndMOVES  }

{doinSP
SPArrivalsJFN := rh jsACs|1

if ~ DCicp then {

    OKGoQ()
    if ~ CheckDC() then {
```

```
        RportL("Waiting to move short-period data")
        resultis false  }

    ScriptJFN := CreateOutput("SWF-D.SCRIPT",7)
    startdc(ScriptJFN)
    RportL("Beginning Datacomputer session")
    senddc(LOGINstr)
    DCicp := true       }

jsACs!1 := POINT (7,CFname)
jsACs!2 := SPArrivalsJFN
jsACs!3 := #000100,,0           // get extension field only
JSYS(jsJFNS,jsACs)

ASCIZToString(CFname,CFnameBCPL)
let rstg := vec 100
append(rstg,"Working from file: ",rstg)
RportL(append(append(rstg,"SP-ARRIVALS.",rstg),CFnameBCPL,rstg))

// construct variable portion of current file, a name of the form Ynnnn.Mnn.Dnn
// from the form Ynnnn%Mnn%Dnn

changesubstr(CFnameBCPL,"%",".")
append(SWF,BasicSWFname,SWF)
append(SWF,CFnameBCPL,SWF)

append(PESF,BasicPESFname,PESF)
append(PESF,CFnameBCPL,PESF)

if debugging then
    {
        SWF := "SWF"
        PESF := "ESF%UNINVERTED"
    }
```

```
MovesNSPF:

// Construct NSPF name

for i := 1 to 50 do NXPF¦i := 0

append(NXPF,BasicNSPFname,NXPF)
append(NXPF,CFnameBCPL,NXPF)

senddc("OPEN ") ; senddc(NXPF) ; senddc(" READ, SYN=NSPF;*c*l")
senddc("OPEN ") ; senddc(PESF) ; senddc(" WRITE, SYN=PESF;*c*l")
senddc("OPEN ") ; senddc(SWF) ; senddc(" APPEND, SYN=SWF;*c*l")
senddc("OPEN PUTS;*n")
opendc(SWFportS,36)

// The following Datalanguage should work for a day's worth of
// NSPF data if the data are not split across a day boundary.
// The request uses one PORT and three FILEs.

senddc("BEGIN   FOR X IN PUTS.ESFPUT*c*l")
senddc("*tUPDATE Y IN PESF WITH Y.EINDEX EQ X.EINDEX*c*l")
senddc("*tUPDATE Z IN Y.ARRIVALS WITH Z.AINDEX EQ X.AINDEX*c*l")
senddc("*tBEGIN          Z.DATASEGSTART=X.DATASEGSTART*c*l")
senddc("*tZ.AMP=0        Z.WAVEFORMAVAIL='Y'        END*c*l")
senddc("APPEND A IN SWF, B IN PUTS.SWFPUT*c*l")
senddc("BEGIN            A.EVENTID=B.EVENTID              A.STA=B.STA*c*l")
senddc("A.CHANTYPE=B.CHANTYPE    A.RATE=B.RATE    A.CHANID=B.CHANID*c*l")
senddc("A.GAIN=B.GAIN   A.COMP=B.COMP   A.START=B.START*c*l")
senddc("A.DATAFORMAT='G'         A.SCALEFACTOR=B.SCALEFACTOR*c*l")
senddc("FOR C IN NSPF WITH C.STINDEX EQ B.STINDEX*c*l")
senddc("FOR D IN C.DATA WITH D.DATE = B.DSDATE
                AND D.TIME GE B.DSTIME AND D.TIME LE B.DETIME*c*l")
senddc("FOR E IN  A.TIMESERIES, F IN D.TIMESERIES*c*l")
senddc("E.DATUM=F.DATUM          END     END;*c*l")
```

```
jsACs¦1 := SPArrivalsJFN
jsACs¦2 := #440000,,#303000
if JSYS(jsOPENF,jsACs) eq failed then {
        RportL("Failure opening SP-Arrivals input file")
        resultis false  }

SPBytesMoved := 0

{sputl
for i := 1 to 512 do A¦i := 0
SIN(SPArrivalsJFN,POINT(36,A),27,$"z)
let BytesSent := puttodc(0,POINT (36,A),-28)

SPBytesMoved := SPBytesMoved + BytesSent

}sputl repeatwhile dcputstate & ~ EofFlg

endputdc()

senddc("CLOSE SWF;CLOSE PESF;CLOSE NSPF;*c*1")

// check SPBytesMoved

RportL("End SP moves")

// Close and delete SP-Arrivals file

CLOSF(SPArrivalsJFN)

}doinSP


senddc("CLOSE %OPEN;*c*1")
```

```
EndMOVES:
// When all done with ARRIVALS file, delete it.

// terminate Datacomputer connection

quitdc()
DCicp := false
EndWrite(ScriptJFN)
resultis false

}moves


// Each ESF day file can have up to 100 events, and up to 999
// arrivals per event (average of 500)

and let CRInput() := valof

{crinp

let failmark := false

RportL("Setting up to move waveforms")

if ~ CRInputL() then failmark := false
if ~ CRInputS() & failmark = false  then resultis false

resultis true

}crinp
```

4.1 CRInputL, Create Long-Period-Copy Driver File

```
and let CRInputL() := valof

{crinpl

// check for ARRIVALS.* Tenex files
// if none, there's no work to do

jsACs¦1 := #001101,,0           // #100101,,0    ?
jsACs¦2 := (POINT7x0,,EventsFiles)

if JSYS (jsGTJFN,jsACs) eq failed then resultis false


let CEventsFileName := vec 50
for i := 1 to 50 do CEventsFileName¦i := 0
EventsJFN := rh jsACs¦1

jsACs¦1 := (POINT7x0,,CFname)
jsACs¦2 := EventsJFN
jsACs¦3 := #000100,,0           // get extension field only
JSYS(jsJFNS,jsACs)

ASCIZToString(CFname,CFnameBCPL)
let rstg := vec 100
append(append(CEventsFileName,"ARRIVALS.",CEventsFileName),CFnameBCPL,CEventsFileName)
append(rstg,"Working from file: ",rstg)
RportL(append(rstg,CEventsFileName,rstg))
```

```
let lstg := vec 50
LoutJFN := CreateOutput(append(append(lstg,"LP-ARRIVALS.",lstg),CFnameBCPL,lstg),36)


jsACs!1 := EventsJFN
jsACs!2 := #440000,,#303000

if JSYS(jsOPENF,jsACs) eq failed then

    {   RportL("Failure opening LP-ARRIVALS file")
        resultis false
    }

// clear ESFPut & SWFPut buffer areas

for i := 1 to csize EsfL do ESFPut!i := 0
for i := 1 to csize SwfL do SWFPut!i := 0


until EofFlg do {loutl
LoutL:

let STimeStr := vec 1
let ETimeStr := vec 1

// clear R (Request Area) & P (debugging Put Area)

for i := 1 to 512 do R!i := 0
for i := 1 to 512 do P!i := 0

SIN(EventsJFN,POINT(36,R),19,$^z)
if EofFlg then endblock loutl

// set up byte pointers to buffers
```

```
Rptr := (POINT7x0,,R)
Pptr := (POINT7x0,,P)

/* The patime (phase arrival time) field is used by SDAC to transmit the
   arrival time of a waveform which is to be copied into the SWF.  The SWF-D
   program computes the window within which timeframe it will attempt to
   locate the segment of long-period data which represents the desired
   waveform.  The left-edge of the window (earliest chronologically) is
   computed by subtracting the value found in the dsdate (datasegment date)
   field from the patime field.  The right-edge of the window is computed
   by adding the value found the the amp (amplitude) field to the patime
   field.
                                                                        */

for c := 1 to 8 do STimeStr>>TD.digit^c := R>>Req.patime^c

CSecInDay := TimetoInt(STimeStr)

if CSecInDay < 0 then

    {   RportL("CRInputL: Bad time-string input")

// gather data sufficient to identify problem request for operator  <<<

        loop loutl
    }

/* Check for waveform being recorded after midnight.  This case needs
   special handling because part of the waveform is in the next day's
   SRO data.
                                                                        */

let padayvec := vec 1
let evdayvec := vec 1
```

```
for c := 2 to 6 do padayvec>>string.c^(c-1) := R>>Req.padate^c
padayvec>>string.n := 5
for c := 1 to 5 do evdayvec>>string.c^c := R>>Req.eventnum^c
evdayvec>>string.n := 5

if ~ Eqstr(padayvec,evdayvec) then loop loutl



/* The dsdate field is used by SDAC as a temporary holding place for the
   amount, in seconds, by which the window should precede the arrival time.
                                                                        */

let NumStg := vec 3
for c := 1 to 6 do NumStg>>string.c^c := R>>Req.dsdate^c
NumStg>>string.n := 6
PrecedingCSeconds := TxtToInt(NumStg)*100


let WhenStart := CSecInDay - PrecedingCSeconds
if WhenStart < 0 then
    {   RportL("Window overlaps previous day")
// set up special input file for overlapping day cases
        loop loutl
    }

if ~ InttoTime(WhenStart,STimeStr) then

    {   RportL("CRInputL: Bad time value")

// need to gather more info for operator intervention

        loop loutl
    }
```

```
// STimeStr holds left-edge window time

/* The amp field is used by SDAC to indicate the number of seconds
   to add to the arrival time to determine the right-edge of the
   window.
                                                                  */

for c := 1 to 7 do NumStg>>string.c^c := R>>Req.amp^c
NumStg>>string.n := 7
FollowingCSeconds := TxtToInt(NumStg)*100

let WhenEnd := (CSecInDay + FollowingCSeconds + 5999)/6000

if WhenEnd > 1439 then
    {   RportL("Window overlaps following day")
// set up special input file for overlapping day cases
        loop loutl
    }


// If not known SRO station, ignore request

{cksta

let wrkvec := vec 2

for c := 1 to 4 do wrkvec>>string.c^c := R>>Req.sta^c
wrkvec>>string.n := 4

for ix := 1 to Stations>>StationData.Stationix do

    {
        if (Eqstr (wrkvec, (lv (Stations>>StationData.Station^ix.SName^1))))
        then endblock cksta
    }
```

```
loop loutl                          // station name not recognized

}cksta

compcount := 1              // initialize # of components to move

switchon R>>Req.comp into

    {
        case $v:
        case $V:                                  // maybe "V" for vertical
        case $z:
        case $Z:        Component := vertical
                        endcase

        case $n:
        case $N:        Component := north
                        endcase

        case $e:
        case $E:        Component := east
                        endcase

        case $a:
        case $A:        Component := all          // vertical, north & east
                        compcount := 3
                        endcase

        default:        loop loutl

    }

if debugging then
```

```
{debug

for lx := 1 to compcount do

{fillup

// ready now to fill PutL structures

FillPutL:

P>>PutL.Esf.EsfCount := P>>PutL.Esf.EsfCount + 1
P>>PutL.Esf.eindex := R>>Req.eindex
P>>PutL.Esf.aindex := R>>Req.aindex

// The ESF file datasegment start date is set from the phase arrival date field

for c := 1 to 6 do P>>PutL.Esf.dsdate^c := R>>Req.padate^c

// The datasegment start time, as computed above, is copied from STimeStr

for c := 1 to 8 do P>>PutL.Esf.dstime^c := STimeStr>>TD.digit^c

P>>PutL.Swf.SwfCount := compcount
if Component = all then Component := vertical     // move first comp

for c := 1 to 5 do P>>PutL.Swf.evdate^c := R>>Req.eventnum^c
for c := 1 to 4 do P>>PutL.Swf.evnum^c := R>>Req.eventnum^(c+5)


for c := 1 to 5 do P>>PutL.Swf.sta^c := R>>Req.sta^c

P>>PutL.Swf.chantype := R>>Req.chantype

for c := 1 to 2 do P>>PutL.Swf.rate^c := R>>Req.rate^c
```

```
for c := 1 to 4 do P>>PutL.Swf.chanid^c := R>>Req.chanid^c

P>>PutL.Swf.gain := R>>Req.gain

P>>PutL.Swf.comp := R>>Req.comp

// The SWF file datasegment start date is set from the phase arrival date field

for c := 1 to 6 do P>>PutL.Swf:dsdate^c := R>>Req.padate^c

// The SWF file datasegment start time is the computed value in STimeStr

for c := 1 to 8 do P>>PutL.Swf.dstime^c := STimeStr>>TD.digit^c


/* The SWF file scalefactor field is set from the ESF file datasegment
   start time field, used by SDAC as a temporary holding place for the
   value.
                                                                    */

for c := 1 to 8 do P>>PutL.Swf.scalefactor^c := R>>Req.dstime^c

for c := 1 to 5 do P>>PutL.Swf.staname^c := R>>Req.sta^c

P>>PutL.Swf.starti := WhenStart/6000

P>>PutL.Swf.endi := WhenEnd

P>>PutL.Swf.typ := Component
Component := Component + 1                  // reflect next component to move
                                            // if all have been requested

if (compcount eq 1 \ (compcount eq all & Component eq 2)) then
```

```
{firstcomponent

SOUT(LoutJFN,POINT(36,P),24)

    {   PrReq(#101)
        PrPutL(#101)
    }

}firstcomponent

}fillup
}debug


unless debugging then

{forreal

FillESF:

let eix := ESFPut>>EsfL.EsfCount + 1

ESFPut>>EsfL.Esf^eix.eindex := R>>Req.eindex
ESFPut>>EsfL.Esf^eix.aindex := R>>Req.aindex

// The ESF file datasegment start date is set from the phase arrival date field

for c := 1 to 6 do ESFPut>>EsfL.Esf^eix.dsdate^c := R>>Req.padate^c

// The datasegment start time, as computed above, is copied from STimeStr

for c := 1 to 8 do ESFPut>>EsfL.Esf^eix.dstime^c := STimeStr>>TD.digit^c

FillSWF:
if Component = all then Component := vertical     // move first comp
```

```
for lx := 1 to compcount do

{fillswf

let six := SWFPut>>SwfL.SwfCount + 1
SWFPut>>SwfL.SwfCount := six

for c := 1 to 5 do SWFPut>>SwfL.Swf^six.evdate^c := R>>Req.eventnum^c
for c := 1 to 4 do SWFPut>>SwfL.Swf^six.evnum^c := R>>Req.eventnum^(c+5)


for c := 1 to 5 do SWFPut>>SwfL.Swf^six.sta^c := R>>Req.sta^c

SWFPut>>SwfL.Swf^six.chantype := R>>Req.chantype

for c := 1 to 2 do SWFPut>>SwfL.Swf^six.rate^c := R>>Req.rate^c

for c := 1 to 4 do SWFPut>>SwfL.Swf^six.chanid^c := R>>Req.chanid^c

SWFPut>>SwfL.Swf^six.gain := R>>Req.gain

SWFPut>>SwfL.Swf^six.comp := R>>Req.comp

// The SWF file datasegment start date is set from the phase arrival date field

for c := 1 to 6 do SWFPut>>SwfL.Swf^six.dsdate^c := R>>Req.padate^c

// The SWF file datasegment start time is the computed value in STimeStr

for c := 1 to 8 do SWFPut>>SwfL.Swf^six.dstime^c := STimeStr>>TD.digit^c


/* The SWF file scalefactor field is set from the ESF file datasegment
   start time field, used by SDAC as a temporary holding place for the
   value.
```

```
                                                                    */

for c := 1 to 8 do SWFPut>>SwfL.Swf~six.scalefactor~c := R>>Req.dstime~c

for c := 1 to 5 do SWFPut>>SwfL.Swf~six.staname~c := R>>Req.sta~c

SWFPut>>SwfL.Swf~six.starti := WhenStart/6000

SWFPut>>SwfL.Swf~six.endi := WhenEnd

SWFPut>>SwfL.Swf~six.typ := Component
Component := Component + 1                  // reflect next component to move
                                            // if all have been requested

}fillswf

}forreal
}loutl repeatuntil EofFlg

CLOSF(EventsJFN)

// if for real, then output ESF & SWF buffers

EndWrite(LoutJFN)

resultis true

}crinpl
```

4.2 CRInputS, Create Short-Period-Copy Driver File

```
and let CRInputS() := valof

{crinps

// check for ARRIVALS.* Tenex files
// if none, there's no work to do

jsACs¦1 := #001101,,0           // #100101,,0    ?
jsACs¦2 := (POINT7x0,,EventsFiles)

if JSYS (jsGTJFN,jsACs) eq failed then resultis false

let CEventsFileName := vec 20
for i := 1 to 20 do CEventsFileName¦i := 0
EventsJFN := rh jsACs¦1

jsACs¦1 := (POINT7x0,,CFname)
jsACs¦2 := EventsJFN
jsACs¦3 := #000100,,0           // get extension field only
JSYS(jsJFNS,jsACs)
ASCIZToString(CFname,CFnameBCPL)
let rstg := vec 100
append(append(CEventsFileName,"ARRIVALS.",CEventsFileName),CFnameBCPL,CEventsFileName)
append(rstg,"Working from file: ",rstg)
RportL(append(rstg,CEventsFileName,rstg))

// extract Ynnnn.Mnn from CfnameBCPL and
// construct SPDET filename of the form %TOP.SDAC.SPDET.Ynnnn.Mnn
```

```
let wrkvec := vec 4
CopyString(CFnameBCPL,wrkvec)
wrkvec>>string.n := 9    // discard day portion of name
changesubstr(wrkvec,"%",".")
append (BasicSPDETname,wrkvec,SPDETname)

let lstg := vec 50
SoutJFN := CreateOutput(append(append(lstg,"SP-ARRIVALS.",lstg),CFnameBCPL,lstg),36)


jsACs!1 := EventsJFN
jsACs!2 := #440000,,#303000

if JSYS(jsOPENF,jsACs) eq failed then

    {   RportL("Failure opening SP-ARRIVALS file")
        resultis false
    }



until EofFlg do {souts
SoutS:

let STimeStr := vec 1
let ETimeStr := vec 1

// clear R (Request Area) & P (Put Area)

for i := 1 to 512 do R!i := 0
for i := 1 to 512 do P!i := 0

SIN(EventsJFN,POINT(36,R),19,$^z)
if EofFlg then endblock souts
```

```
// set up byte pointers to buffers

Rptr := (POINT7x0,,R)
Pptr := (POINT7x0,,P)

/* The patime (phase arrival time) field transmits the arrival time
   of a waveform which is to be copied into the SWF.  The SWF-D
   program will attempt to locate the segment of short-period data
   which represents the desired waveform.
                                                                  */

for c := 1 to 8 do STimeStr>>TD.digit"c := R>>Req.patime"c

CSecInDay := TimetoInt(STimeStr)

if CSecInDay < 0 then

    {   RportL("CRInputS: Bad time-string input")

// gather data sufficient to identify problem request for operator  <<<

        loop souts
    }

/* Check for waveform being recorded after midnight.  This case needs
   special handling because part of the waveform is in the next day's
   SRO data.
                                                                  */

let padayvec := vec 1
let evdayvec := vec 1

for c := 2 to 6 do padayvec>>string.c"(c-1) := R>>Req.padate"c
padayvec>>string.n := 5
for c := 1 to 5 do evdayvec>>string.c"c := R>>Req.eventnum"c
evdayvec>>string.n := 5
```

```
if ~ Eqstr(padayvec,evdayvec) then loop souts


/* The dsdate field is used by SDAC as a temporary holding place for the
   amount, in seconds, by which the window should precede the arrival time.
                                                                    */

let NumStg := vec 3
for c := 1 to 6 do NumStg>>string.c"c := R>>Req.dsdate"c
NumStg>>string.n := 6
PrecedingCSeconds := TxtToInt(NumStg)*100


let WhenStart := CSecInDay - PrecedingCSeconds
if WhenStart < 0 then
    {   RportL("Window overlaps previous day")
// set up special input file for overlapping day cases
        loop souts
    }

if ~ InttoTime(WhenStart,STimeStr) then

    {   RportL("CRInputS: Bad time value")

// need to gather more info for operator intervention

        loop souts
    }

// STimeStr holds left-edge window time

/* The amp field is used by SDAC to indicate the number of seconds
   to add to the arrival time to determine the right-edge of the
   window.
                                                                    */
```

```
for c := 1 to 7 do NumStg>>string.c~c := R>>Req.amp~c
NumStg>>string.n := 7
FollowingCSeconds := TxtToInt(NumStg)*100

let WhenEnd := (CSecInDay + FollowingCSeconds + 5999)/6000

if WhenEnd > 1439 then
    {   RportL("Window overlaps following day")
// set up special input file for overlapping day cases
        loop souts
    }


{cksta

// If not known SRO station, ignore request

let wrkvec := vec 2

for c := 1 to 4 do wrkvec>>string.c~c := R>>Req.sta~c
wrkvec>>string.n := 4

for ix := 1 to Stations>>StationData.Stationix do

    {
        if (Eqstr (wrkvec, (lv (Stations>>StationData.Station~ix.SName~1))))
        then endblock cksta
    }

loop souts                          // station name not recognized

}cksta


if debugging then
```

```
{debug

{fillup

// ready now to fill PutS structures

FillPutS:

P>>PutS.Esf.EsfCount := P>>PutS.Esf.EsfCount + 1
P>>PutS.Esf.eindex := R>>Req.eindex
P>>PutS.Esf.aindex := R>>Req.aindex

// The ESF file datasegment start date is set from the phase arrival date field

for c := 1 to 6 do P>>PutS.Esf.dsdate^c := R>>Req.padate^c

// The datasegment start time, as computed above, is copied from STimeStr.
// The time may be adjusted by the SPThere routine after inspection of the
// SPDET file.


for c := 1 to 8 do P>>PutS.Esf.dstime^c := STimeStr>>TD.digit^c

P>>PutS.Swf.SwfCount := P>>PutS.Swf.SwfCount + 1

for c := 1 to 5 do P>>PutS.Swf.evdate^c := R>>Req.eventnum^c
for c := 1 to 4 do P>>PutS.Swf.evnum^c := R>>Req.eventnum^(c+5)


for c := 1 to 5 do P>>PutS.Swf.sta^c := R>>Req.sta^c


P>>PutS.Swf.chantype := R>>Req.chantype
```

```
for c := 1 to 2 do P>>PutS.Swf.rate^c := R>>Req.rate^c

for c := 1 to 4 do P>>PutS.Swf.chanid^c := R>>Req.chanid^c

P>>PutS.Swf.gain := R>>Req.gain

P>>PutS.Swf.comp := R>>Req.comp

// The SWF file datasegment start date is set from the phase arrival date field

for c := 1 to 6 do P>>PutS.Swf.dsdate^c := R>>Req.padate^c
for c := 1 to 6 do P>>PutS.Swf.DSdate^c := R>>Req.padate^c

// The SWF file datasegment start time is the computed value in STimeStr

for c := 1 to 8 do P>>PutS.Swf.dstime^c := STimeStr>>TD.digit^c


/* The SWF file scalefactor field is set from the ESF file datasegment
   start time field, used by SDAC as a temporary holding place for the
   value.
                                                                    */

for c := 1 to 8 do P>>PutS.Swf.scalefactor^c := R>>Req.dstime^c


}fillup
}debug


unless debugging then

{forreal
```

```
FillESF:

let eix := ESFPut>>EsfL
ESFPut>>EsfL.Esf^eix.eindex := R>>Req.eindex
ESFPut>>EsfL.Esf^eix.aindex := R>>Req.aindex

// The ESF file datasegment start date is set from the phase arrival date field

for c := 1 to 6 do ESFPut>>EsfL.Esf^eix.dsdate^c := R>>Req.padate^c

// The datasegment start time, as computed above, is copied from STimeStr

// for c := 1 to 8 do ESFPut>>EsfL.Esf^eix.dstime^c := STimeStr>>TD.digit^c

FillSWF:

for lx := 1 to compcount do

{fillswf

let six := SWFPut>>SwfL.SwfCount + 1
SWFPut>>SwfL.SwfCount := six

for c := 1 to 5 do SWFPut>>SwfL.Swf^six.evdate^c := R>>Req.eventnum^c
for c := 1 to 4 do SWFPut>>SwfL.Swf^six.evnum^c := R>>Req.eventnum^(c+5)


for c := 1 to 5 do SWFPut>>SwfL.Swf^six.sta^c := R>>Req.sta^c

SWFPut>>SwfL.Swf^six.chantype := R>>Req.chantype

for c := 1 to 2 do SWFPut>>SwfL.Swf^six.rate^c := R>>Req.rate^c

for c := 1 to 4 do SWFPut>>SwfL.Swf^six.chanid^c := R>>Req.chanid^c
```

```
SWFPut>>SwfL.Swf~six.gain := R>>Req.gain

SWFPut>>SwfL.Swf~six.comp := R>>Req.comp

// The SWF file datasegment start date is set from the phase arrival date field

for c := 1 to 6 do SWFPut>>SwfL.Swf~six.dsdate~c := R>>Req.padate~c

// The SWF file datasegment start time is the computed value in STimeStr

// for c := 1 to 8 do SWFPut>>SwfL.Swf~six.dstime~c := STimeStr>>TD.digit~c


/* The SWF file scalefactor field is set from the ESF file datasegment
   start time field, used by SDAC as a temporary holding place for the
   value.
                                                                      */

for c := 1 to 8 do SWFPut>>SwfL.Swf~six.scalefactor~c := R>>Req.dstime~c

// for c := 1 to 5 do SWFPut>>SwfL.Swf~six.staname~c := R>>Req.sta~c

// SWFPut>>SwfL.Swf~six.starti := WhenStart/6000

// SWFPut>>SwfL.Swf~six.endi := WhenEnd

SWFPut>>SwfL.Swf~six.typ := Component
}fillswf

}forreal

for c := 1 to 8 do P>>PutS.Swf.scalefactor~c := R>>Req.dstime~c

if SPThere() then {
 RportL("Short period data")
```

```
}

// SOUT(FailJFN,POINT(36,R),19)                    // from Req area

}souts
CLOSF(EventsJFN)
EndWrite(SoutJFN)
// EndWrite(FailJFN)

resultis true

}crinps
```

4.3 SPThere, Check Short-Period Detections Map

```
// SPThere fills station index, detection date, start time
// and end time if check of SPDET
// file indicates that data ought to be available.

and let SPThere() := valof

{spthere

if ~ DCicp then {

    OKGoQ()
    if ~ CheckDC() then {

        RportL("Waiting to check SPDET data")
        resultis false  }

    ScriptJFN := CreateOutput("SWF-D.SCRIPT",7)
    startdc(ScriptJFN)
    RportL("Beginning Datacomputer session")
    senddc(LOGINstr)
    DCicp := true       }

if ~ SPDETopen then {openspdet

senddc("OPEN ") ; senddc(SPDETname) ; senddc(" READ, SYN = SPDET;*c*l")

if ~ DCLook() then resultis false
SPDETopen := true
```

```
senddc("OPEN %TOP.SDAC.CCA.SWF.SSPDET, SYN = SPDETP;*c*1")
opendc("SPDETP",36)

}openspdet


let padayvec, psdayvec, pedayvec, padayn := vec 1, vec 1, vec 1, nil
for c := 2 to 6 do padayvec>>string.c¨(c-1) := R>>Req.padate¨c
padayvec>>string.n := 5

padayn := TxtToInt(padayvec)
inttotxt(padayn-1,psdayvec)
inttotxt(padayn+1,pedayvec)

let pstimevec := vec 3
for c := 1 to 8 do pstimevec>>string.c¨c := P>>PutS.Swf.dstime¨c
pstimevec>>string.n := 8

let petimevec := vec 3
for c := 1 to 8 do petimevec>>string.c¨c := P>>PutS.Swf.dstime¨c// <<<fix time
petimevec>>string.n := 8

let stavec := vec 2
for c := 1 to 4 do stavec>>string.c¨c := R>>Req.sta¨c
stavec>>string.n := 4

// senddc("SPDETP = SPDET WITH SDATE GE ") ; senddc(psdayvec)
senddc("SPDETP = SPDET WITH SDATE = ") ; senddc(padayvec)
// senddc(" AND SDATE LE ") ; senddc(pedayvec)
senddc(" AND STA = '") ; senddc(stavec)
senddc("';*c*1")

let sumdets := 0
{moredets
let detBUFF := vec 50
let detPTR := POINT (36,detBUFF)
```

```
let spdets := getfromdc(0,detPTR,5,$^z)
if spdets eq 0 then break moredets
sumdets := sumdets + spdets
// pick out detection and fill PutS structure

P>>PutS.Swf.stindex := detBUFF>>SSPDET.standex

for c := 1 to 8 do P>>PutS.Swf.DStime^c := detBUFF>>SSPDET.stime^c
for c := 1 to 8 do P>>PutS.Swf.dstime^c := detBUFF>>SSPDET.stime^c
for c := 1 to 8 do P>>PutS.Esf.dstime^c := detBUFF>>SSPDET.stime^c

for c := 1 to 8 do P>>PutS.Swf.DEtime^c := detBUFF>>SSPDET.etime^c

if debugging then    {   PrReq(#101);   PrPutS(#101)    }
      SOUT(SoutJFN,POINT(36,P),28)
}moredets repeatwhile dcgetstate

if sumdets eq 0 then resultis false     // no detections for that day

if ~ DCLook() then {problem

    RportL("Trouble with SPDET")

senddc("CLOSE %OPEN;*c*l")

RportL("Ending Datacomputer session")
quitdc()

EndWrite(ScriptJFN)
DCicp := false
resultis false
}problem
resultis true
}spthere
```

5. The SPDET File Generator

```
// SWF-D Program:  GAvail Module

// GAvail constructs and maintains a map of the short-period SRO
// detections on the Datacomputer.  It is by means of this information
// that we are able to determine whether requested waveforms may be
// expected to be available.  The data are constructed as monthly
// files under the %TOP.SDAC.VELANET.SPDET node with the year and month
// used as pathname keys to the period covered.  The complete pathname
// is of the form: %TOP.SDAC.VELANET.SPDET.Ynnnn.Mnn -- and the first
// file generated is for January, 1978.

get     "<CCA-SWF>SWFHEAD.BCP"

external        {       CheckDC         }
external        {       CheckL          }
external        {       DCLook          }
external        {       GAvail          }
external        {       MARK            }
external        {       OKGoQ           }
external        {       RportL          }

static  {stat

 Tick           :       nil
 Basicpath      :       "%TOP.SDAC.VELANET."
 LOGINstr       :       "Login SDAC.CCA.SWF;*c*1"
 spdetNAME      :       vec     100
 nspfFRAG       :       vec     100
 nspfNAME       :       vec     100
```

```
 DCnspfNAME      :       vec     100
 MsgsBCPL        :       vec     512
 MsgsBuffer      :       vec     512
 GAyear          :       "0000"
 GAmonth         :       "00"
 GAday           :       "00"
 GAYint          :       nil
 GAMint          :       nil
 GADint          :       nil
 ndays           :       nil
 moredetections  :       nil
 jsACs           :       vec     10

}stat

let GAvail() := valof



{GAvail

MARK(TaskStatus,GeneratingSegMap)

Tick := 0
let sdp := lv Stations>>StationData.AllStationsSPDETDate
let adp := lv Stations>>StationData.AllStationsASLDate

// Adjust for valid next short-period file date


jsACs¦2 := (sdp>>Date.Yr),,((sdp>>Date.Mo) - 1)              // JAN = 0
jsACs¦3 := ((sdp>>Date.Day) - 1),,0                      // Day 1 = 0
jsACs¦4 := 0,,5                                  // 5 seconds past midnight
                                                 // to ensure right day
```

```
if JSYS(jsIDCNV,jsACs) eq failed then {fixdate

sdp>>Date.Day := 1
sdp>>Date.Mo := sdp>>Date.Mo + 1

if sdp>>Date.Mo > 12 then {hnewyr

sdp>>Date.Yr := sdp>>Date.Yr + 1
sdp>>Date.Mo := 1

}hnewyr

}fixdate

GAYint := sdp>>Date.Yr
GAMint := sdp>>Date.Mo
GADint := sdp>>Date.Day

// Check for how much work there is to do.

let DeltaYear := adp>>Date.Yr - sdp>>Date.Yr
let DeltaMonth := adp>>Date.Mo - sdp>>Date.Mo
if DeltaYear > 0 then DeltaMonth := DeltaMonth + 12
if DeltaMonth ge 1 then {
    ndays := ((DaysPerMonth|(sdp>>Date.Mo)) - sdp>>Date.Day) + 1
    goto DoSPD
}
ndays := adp>>Date.Day - sdp>>Date.Day

if ndays > 0 then goto DoSPD

NoWork:

MARK(TaskStatus,EndGenSegMap)
RportL("SPDET files are up-to-date")
resultis true
```

```
DoSPD:

RportL("Generating segment availability map")

// Check whether Datacomputer session is already in progress -
// and if not, initiate the connection.

if ~ DCicp then {

    OKGoQ()
    if ~ CheckDC() then {

        RportL("Waiting to generate segment availability map")
        resultis false  }

    ScriptJFN := CreateOutput("SWF-D.SCRIPT",7)
    startdc(ScriptJFN)
    RportL("Beginning Datacomputer session")
    senddc(LOGINstr)
    DCicp := true       }


inttotxt (GAYint, GAyear)       // integer to text conversions
inttotxt (GAMint, GAmonth)
inttotxt (GADint, GAday)

let rstg := vec 100
append(rstg,"Starting day.mo.yr = ",rstg)
append(rstg,GAday,rstg) ; addch($.,rstg)
append(rstg,GAmonth,rstg)
append(addch($.,rstg),GAyear,rstg) ; append(rstg," for ",rstg)
let nstg := vec 5
RportL(append(append(rstg,inttotxt(ndays,nstg),rstg)," days",rstg))

// Construct detections file name
```

```
for i := 0 to 100 do spdetNAME¦i := 0
for i := 0 to 100 do nspfFRAG¦i := 0
addch($Y,nspfFRAG)
append(nspfFRAG,GAyear,nspfFRAG)
append(nspfFRAG,".M",nspfFRAG)
if GAMint < 10 then { addch($0,nspfFRAG)  }
append(nspfFRAG,GAmonth,nspfFRAG)

append(spdetNAME,Basicpath,spdetNAME)
append(spdetNAME,"SPDET.",spdetNAME)

// append(spdetNAME,"%TOP.SDAC.CCA.SWF.TEST%SPDET.",spdetNAME)  // for debugging

append(spdetNAME,nspfFRAG,spdetNAME)



// If the starting day = 1, (new month) then create a new SPDET file.

if GADint eq 1 then ¦nu

    senddc("DELETE ") ; senddc(spdetNAME) ; senddc(";*c*1")
    senddc("CREATE ") ; senddc(spdetNAME)
    senddc(" FILE LIKE %TOP.SDAC.VELANET.PROTOTYPES.SPDET;*c*1")

// Then close it.

senddc("CLOSE ") ; senddc(spdetNAME) ; senddc(";*c*1")
let rstg := vec 100
append(rstg,"Created ",rstg)
RportL(append(rstg,spdetNAME,rstg))

¦nu

// Open LIST1, a Datacomputer file used only for Datalanguage loop control.
```

```
senddc("OPEN %TOP.SDAC.VELANET.PROTOTYPES.LIST1;*c*l")

// Construct nspf day filename root

for i := 0 to 100 do nspfNAME¦i := 0
append(nspfNAME,Basicpath,nspfNAME)
append(nspfNAME,"NSPF.",nspfNAME)
append(nspfNAME,nspfFRAG,nspfNAME)



// Ready now to loop through day files.

for api := GADint to (GADint + ndays - 1) by 1 do {aplp

// Construct specific day filename

append(nspfNAME,".D",DCnspfNAME)
if GADint < 10 then { addch($0,DCnspfNAME)  }
append(DCnspfNAME,GAday,DCnspfNAME)
GADint := GADint + 1
inttotxt(GADint, GAday)

// Check that file exists

senddc("LIST ") ; senddc(DCnspfNAME) ; senddc(";*c*l")

if ~ DCLook() then { moredetections := false ;  break aplp }

// Open day file:  OPEN %TOP.SDAC.VELANET.NSPF.Ynnnn.Mnn.Dnn,SYN=SPF;

senddc("OPEN ") ; senddc(DCnspfNAME) ; senddc(", SYN = SPF;*c*l")
if ~ DCLook() then { moredetections := false ;  break aplp }

// Check that file is on-line
```

```
senddc("LIST SPF %STATUS;*c*1")
while dcgetstate do getfromdc (0,POINT(7,MsgsBuffer),-2560)
// search substring for "online" status
ASCIZToString(MsgsBuffer,MsgsBCPL)
let stch,endch := nil,nil
if ~ findsubstr(MsgsBCPL,"STAT=ONLINE",lv stch,lv endch,1) then {offday

    let rstg := vec 100
    append(rstg,"Off-line file:  ",rstg)
    RportL(append(rstg,DCnspfNAME))
    break aplp                          // option here to continue instead

}offday


// Open detections file

senddc("OPEN ") ; senddc(spdetNAME)
senddc(" APPEND DEFER, SYN = SPDET;*c*1")
if ~ DCLook() then break aplp   // cannot proceed if there are file problems



// Send Datalanguage to transfer detections from day file to detection file

scriptdc(0)             // inhibit scripting

senddc("BEGIN DECLARE F INT F=1*c*1")
senddc("UNTIL F<0 DO BEGIN*c*1")
senddc("FOR SPF WITH FLAG EQ F AND STA NE *'XXXXX*' BEGIN*c*1")
senddc("*tDECLARE ODATE INT DECLARE OTIME INT DECLARE ODEX INT*c*1")
senddc("*tDECLARE PDATE INT DECLARE PTIME INT*c*1")
senddc("*tDECLARE CSTADEX INT DECLARE CCOUNT INT*c*1")
senddc("*tDECLARE CSTA STR(5) CSTA=STA*c*1")
senddc("*tCSTADEX=STINDEX CCOUNT=COUNT*c*1")
senddc("FOR DATA BEGIN*c*1")
```

```
senddc("*tIF INDEX EQ 1 THEN*c*l")
senddc("*tBEGIN ODATE=DATE OTIME=TIME ODEX=1 END*c*l")
senddc("ELSE IF DET EQ 1 THEN*c*l")
senddc("FOR SPDET,LIST1 BEGIN *c*l")
senddc("*tSTA=CSTA STANDEX=CSTADEX*c*l")
senddc("*tSDATE=ODATE STIME=OTIME SINDEX=ODEX*c*l")
senddc("*tEDATE=PDATE ETIME=PTIME EINDEX=INDEX-1*c*l")
senddc("*tODATE=DATE OTIME=TIME ODEX=INDEX END*c*l")
senddc("*tPDATE=DATE PTIME=TIME END*c*l")
senddc("FOR SPDET,LIST1 BEGIN*c*l")
senddc("*tSTA=CSTA STANDEX=CSTADEX*c*l")
senddc("*tSDATE=ODATE STIME=OTIME SINDEX=ODEX*c*l")
senddc("*tEDATE=PDATE ETIME=PTIME EINDEX=CCOUNT END*c*l")
senddc("END F=F-1 END END;*c*l")

scriptdc(ScriptJFN)              // resume scripting

// Close both detections and NSPF files

senddc("CLOSE SPF; CLOSE SPDET;*c*l")

let rstg := vec 100
RportL(append(append(rstg,DCnspfNAME,rstg)," [OK]",rstg))
Stations>>StationData.AllStationsSPDETDate.Yr := GAYint
Stations>>StationData.AllStationsSPDETDate.Mo := GAMint
Stations>>StationData.AllStationsSPDETDate.Day := GADint - 1
MARK(StationsStatus,Stations>>StationData.AllStationsSPDETDate)

// Quit voluntarily between processing of NSPF day files if load average
// is high or Datacomputer is busy.

//

Tick := Tick + 1
```

```
if Tick < 3 then loop aplp
Tick := 0
if ~ CheckL() then { moredetections := true ; goto EndSPD }
if ~ CheckDC() then { moredetections := true ; goto EndSPD }

}aplp



// Create Tenex file for detections <<< not doing it yet

// {outf
//
// let rstg := vec 100
// append(append(rstg,"SPDET.",rstg),changesubstr(nspfFRAG,".","%"),rstg)
// let detJFN := CreateOutput(rstg,36)
//
// senddc("OPEN ") ; senddc(spdetNAME) ; senddc(" READ, SYN = SPDET;*c*l")
// senddc("OPEN %TOP.SDAC.VELANET.PROTOTYPES.SPDETP, SYN = SPDETP;*c*l")
// opendc("SPDETP",7)          <<< define new port
// senddc("SPDETP = SPDET;*c*l")
//
// {moredets
//
// let detBUFF := vec 512
// let detPTR := POINT(36,detBUFF)
// let spdets := getfromdc(0,detPTR,512,$^z)
// if spdets eq 0 then break moredets
// SOUT(detJFN,detPTR,512,$^z)
//
// }moredets repeatwhile dcgetstate
// CLOSF(detJFN)
// RportL(append(rstg,"*screated",rstg))
// }outf
```

```
// End Datacomputer session

EndSPD:
// Get status of detection file

senddc("LIST ") ; senddc(spdetNAME) ; senddc(" %INFO;*c*1")
if ~ DCLook() then {

    RportL("Trouble with SPDET")  }

senddc("CLOSE %OPEN;*c*1")
RportL("Ending Datacomputer session")
quitdc()

EndWrite(ScriptJFN)
DCicp := false

// set up starting SPDET date for next cycle

let sdp := lv Stations>>StationData.AllStationsSPDETDate
sdp>>Date.Day := sdp>>Date.Day + 1                    // OK if it goes over
                                                      // month boundary
MARK(StationsStatus,sdp>>Date)

resultis ~ moredetections

}GAvail
```

5.1 DCLook, Check Messages from Datacomputer

```
// DCLook returns true¦false according as any Datacomputer messages
// are not¦are indicative of errors

and let DCLook() := valof

{dclook

while dcgetstate do {mlp

let Msgsptr := POINT(7,MsgsBuffer)
let mbytes := getfromdc(0,Msgsptr,512,$*1)

if mbytes eq 0 then loop mlp

let mch := ILDB(lv Msgsptr)

switchon mch into {mchck

  case $- :                              // #55
  case $+ :                              // #53
  case $? :                              // #77
  case $*" :                             // #42
  case $*' :    resultis false           // #56

}mchck
}mlp repeatwhile dcgetstate
resultis true
}dclook
```

6. The Utility Programs Module

```
// SWF-D Program: SWUtil Module
// Contains utility & debugging display routines

get     "<CCA-SWF>SWFHEAD.BCP"

external        {       PrReq           }
external        {       PrPutL          }
external        {       PrPutS          }
external        {       RportL          }


static  {stat

 dJFN   :       #101

}stat
```

6.1 TimetoInt, Convert Time from ASCII String to Integer Value

```
/* TimetoInt converts a string of 8 ASCII digits into an integer.
   Tptr must be the address of the first of 2 words containing 8 9-bit
   ASCII digits, taken to be in the format HHMMSSCC. code tells the
   units of the returned value.
                                                                    */

let TimetoInt(Tptr,code) := valof

{timetoint

if numbargs < 2 then code := CSecsCode

let TmpS := vec 1
for d := 1 to 8 do

    {checkloop

        let nch := Tptr>>TD.digit^d
        if nch < $0 \ nch > $9 then

            {ierr

                RportL("ERROR: TimetoInt found non-numeric data!")

// option here to quit or to construct defaults

                nch := $0
```

```
            }ierr

        TmpS>>TD.digit^d := nch - #60

    }checkloop

let hours := (TmpS>>TD.digit^1)*10 + TmpS>>TD.digit^2
if hours > 23 then { RportL("ERROR: TimetoInt Hours > 23") ; hours := 12 }

let minutes := (TmpS>>TD.digit^3)*10 + TmpS>>TD.digit^4
if minutes > 59 then { RportL("ERROR: TimetoInt Minutes > 59") ; minutes := 0 }

let seconds := (TmpS>>TD.digit^5)*10 + TmpS>>TD.digit^6
if seconds > 59 then { RportL("ERROR: TimetoInt Seconds > 59") ; seconds := 0 }

let csecs := (TmpS>>TD.digit^7)*10 + TmpS>>TD.digit^8

let val := ((hours*60+minutes)*60+seconds)*100+csecs   // val is in centiseconds

switchon code into

    {scaleoutput

        case HoursCode: val := val/60

        case MinutesCode:       val := val/60

        case SecondsCode:       val := val/100

    }scaleoutput

resultis val

}timetoint
```

6.2 InttoTime, Convert Integer into Selected Time Units

```
/* InttoTime takes 3 arguments:
        (1) val - an integer
        (2) Tptr - location of at least 2 words of space to put the output into
        (3) code - tells how to interpret val:
                3 => val is a number of hours
                2 => val is a number of minutes
                1 => val is a number of seconds
                0 => val is a number of centiseconds
                                                                    */

let InttoTime(val,Tptr,code) := valof

{inttotime

if val < 0 resultis false

let hours,minutes,seconds,csecs := 0,0,0,0
let result := true
if numbargs < 3 then code := CSecsCode

switchon code into

    {whereput

        case CSecsCode: csecs := val ; endcase

        case SecondsCode:       seconds := val ; endcase
```

```
        case MinutesCode:       minutes := val ; endcase

        case HoursCode:         hours := val ; endcase

    }whereput

if csecs > 99 then { seconds := csecs/100 ; csecs := csecs rem 100 }

if seconds > 59 then { minutes := seconds/60 ; seconds := seconds rem 60 }

if minutes > 59 then { hours := minutes/60 ; minutes := minutes rem 60 }

if hours > 23 then

    {ierr

        RportL("ERROR: InttoTime found bad input time")

// option here to quit or to continue with default values

        hours := 23
        minutes := 59
        seconds := 59
        csecs := 99
        result := false

    }ierr
```

```
Tptr>>TD.digit^1 := (hours/10) + #60
Tptr>>TD.digit^2 := (hours rem 10) + #60
Tptr>>TD.digit^3 := (minutes/10) + #60
Tptr>>TD.digit^4 := (minutes rem 10) + #60
Tptr>>TD.digit^5 := (seconds/10) + #60
Tptr>>TD.digit^6 := (seconds rem 10) + #60
Tptr>>TD.digit^7 := (csecs/10) + #60
Tptr>>TD.digit^8 := (csecs rem 10) + #60

resultis result

}inttotime
```

6.3 Print Routines

```
// Utility print routines

//  Print <string> = <value>

let MOStr ( dJFN,string,value) be

{mostr
        WriteS(dJFN,string);    WriteS(dJFN," = ")
        WriteS(dJFN,value);     Writech(dJFN,$*n)
}mostr


//  Print <string>(<index>) = <string>

let MOStix ( dJFN,string,index,value) be

{mostix
        WriteS(dJFN,string);    WriteS(dJFN,"("); WriteN(dJFN,index)
        WriteS(dJFN,") = ");    WriteS(dJFN,value); Writech(dJFN,$*n)
}mostix
```

```
//  Print <string> = <octal number>

let MONum ( dJFN,string,octnum) be

{octnum
        WriteS(dJFN,string);  WriteS(dJFN," = #");  WriteOct(dJFN,octnum)
        Writech(dJFN,$*n)
}octnum


//  Print <string> = <decimal number>

let MODec(dJFN,string,decnum) be

{decnum

        WriteS(dJFN,string);  WriteS(dJFN," = ")
        WriteN(dJFN,decnum);  Writech(dJFN,$*n)
}decnum
```

6.4 PrReq, Display Req Structure

```
// PrReq * Display Req structure

let PrReq(dJFN) be

{prreq

WriteS(dJFN,"*nReq --*n")
MODec (dJFN,"EINDEX",R>>Req.eindex)
WriteS(dJFN,"EVENTNUM = ")
for c := 1 to 9 do Writech(dJFN,R>>Req.eventnum^c)
Writech(dJFN,$*n)
MODec (dJFN,"AINDEX",R>>Req.aindex)
WriteS(dJFN,"STA = ")
for c := 1 to 5 do Writech(dJFN,R>>Req.sta^c)
Writech(dJFN,$*n)
WriteS(dJFN,"CHANTYPE = ") ; Writech(R>>Req.chantype)
Writech(dJFN,$*n)
WriteS(dJFN,"RATE = ")
for c := 1 to 2 do Writech(dJFN,R>>Req.rate^c)
Writech(dJFN,$*n)
WriteS(dJFN,"CHANID = ")
for c := 1 to 4 do Writech(dJFN,R>>Req.chanid^c)
Writech(dJFN,$*n)
WriteS(dJFN,"GAIN = ") ; Writech(R>>Req.gain)
Writech(dJFN,$*n)
WriteS(dJFN,"COMP = ") ; Writech(R>>Req.comp)
Writech(dJFN,$*n)
WriteS(dJFN,"DSDATE = ")
for c := 1 to 6 do Writech(dJFN,R>>Req.dsdate^c)
```

```
Writech(dJFN,$*n)
WriteS(dJFN,"DSTIME = ")
for c := 1 to 8 do Writech(dJFN,R>>Req.dstime^c)
Writech(dJFN,$*n)
WriteS(dJFN,"PADATE = ")
for c := 1 to 6 do Writech(dJFN,R>>Req.padate^c)
Writech(dJFN,$*n)
WriteS(dJFN,"PATIME = ")
for c := 1 to 8 do Writech(dJFN,R>>Req.patime^c)
Writech(dJFN,$*n)
WriteS(dJFN,"PHASEID = ")
for c := 1 to 6 do Writech(dJFN,R>>Req.phaseid^c)
Writech(dJFN,$*n)
WriteS(dJFN,"AMP = ")
for c := 1 to 7 do Writech(dJFN,R>>Req.amp^c)
Writech(dJFN,$*n)

}prreq
```

6.5 PrPutL, Display PutL Structure

```
// PrPutL * Display PutL structure

let PrPutL(dJFN) be

{prputl

WriteS(dJFN,"*nPutL --*n")
MODec (dJFN,"EsfCount",P>>PutL.Esf.EsfCount)
MODec (dJFN,"EINDEX",P>>PutL.Esf.eindex)
MODec (dJFN,"AINDEX",P>>PutL.Esf.aindex)
WriteS(dJFN,"DSDATE = ")
for c := 1 to 6 do Writech(dJFN,P>>PutL.Esf.dsdate^c)
Writech(dJFN,$*n)
WriteS(dJFN,"DSTIME = ")
for c := 1 to 8 do Writech(dJFN,P>>PutL.Esf.dstime^c)
Writech(dJFN,$*n)
MODec (dJFN,"SwfCount",P>>PutL.Swf.SwfCount)
WriteS(dJFN,"EVDATE = ")
for c := 1 to 5 do Writech(dJFN,P>>PutL.Swf.evdate^c)
Writech(dJFN,$*n)
WriteS(dJFN,"EVNUM = ")
for c := 1 to 4 do Writech(dJFN,P>>PutL.Swf.evnum^c)
Writech(dJFN,$*n)
WriteS(dJFN,"STA = ")
for c := 1 to 5 do Writech(dJFN,P>>PutL.Swf.sta^c)
Writech(dJFN,$*n)
WriteS(dJFN,"CHANTYPE = ") ; Writech(P>>PutL.Swf.chantype)
Writech(dJFN,$*n)
WriteS(dJFN,"RATE = ")
```

```
for c := 1 to 2 do Writech(dJFN,P>>PutL.Swf.rate^c)
Writech(dJFN,$*n)
WriteS(dJFN,"CHANID = ")
for c := 1 to 4 do Writech(dJFN,P>>PutL.Swf.chanid^c)
Writech(dJFN,$*n)
WriteS(dJFN,"GAIN = ") ; Writech(P>>PutL.Swf.gain)
Writech(dJFN,$*n)
WriteS(dJFN,"COMP = ") ; Writech(P>>PutL.Swf.comp)
Writech(dJFN,$*n)
WriteS(dJFN,"DSDATE = ")
for c := 1 to 6 do Writech(dJFN,P>>PutL.Swf.dsdate^c)
Writech(dJFN,$*n)
WriteS(dJFN,"DSTIME = ")
for c := 1 to 8 do Writech(dJFN,P>>PutL.Swf.dstime^c)
Writech(dJFN,$*n)
WriteS(dJFN,"SCALEFACTOR = ")
for c := 1 to 8 do Writech(dJFN,P>>PutL.Swf.scalefactor^c)
Writech(dJFN,$*n)
WriteS(dJFN,"STANAME = ")
for c := 1 to 5 do Writech(dJFN,P>>PutL.Swf.staname^c)
Writech(dJFN,$*n)
MODec (dJFN,"STARTI",P>>PutL.Swf.starti)
MODec (dJFN,"ENDI",P>>PutL.Swf.endi)
MODec(dJFN,"TYP",P>>PutL.Swf.typ)
Writech(dJFN,$*n)
}prputl
```

6.6 PrPutS, Display PutS Structure

```
// PrPutS * Display PutS structure

let PrPutS(dJFN) be

{prputs

WriteS(dJFN,"*nPutS --*n")
MODec (dJFN,"EsfCount",P>>PutS.Esf.EsfCount)
MODec (dJFN,"EINDEX",P>>PutS.Esf.eindex)
MODec (dJFN,"AINDEX",P>>PutS.Esf.aindex)
WriteS(dJFN,"DSDATE = ")
for c := 1 to 6 do Writech(dJFN,P>>PutS.Esf.dsdate^c)
Writech(dJFN,$*n)
WriteS(dJFN,"DSTIME = ")
for c := 1 to 8 do Writech(dJFN,P>>PutS.Esf.dstime^c)
Writech(dJFN,$*n)
MODec (dJFN,"SwfCount",P>>PutL.Swf.SwfCount)
WriteS(dJFN,"EVDATE = ")
for c := 1 to 5 do Writech(dJFN,P>>PutS.Swf.evdate^c)
Writech(dJFN,$*n)
WriteS(dJFN,"EVNUM = ")
for c := 1 to 4 do Writech(dJFN,P>>PutS.Swf.evnum^c)
Writech(dJFN,$*n)
WriteS(dJFN,"STA = ")
for c := 1 to 5 do Writech(dJFN,P>>PutS.Swf.sta^c)
Writech(dJFN,$*n)
WriteS(dJFN,"CHANTYPE = ") ; Writech(P>>PutS.Swf.chantype)
Writech(dJFN,$*n)
WriteS(dJFN,"RATE = ")
```

```
for c := 1 to 2 do Writech(dJFN,P>>PutS.Swf.rate~c)
Writech(dJFN,$*n)
WriteS(dJFN,"CHANID = ")
for c := 1 to 4 do Writech(dJFN,P>>PutS.Swf.chanid~c)
Writech(dJFN,$*n)
WriteS(dJFN,"GAIN = ") ; Writech(P>>PutS.Swf.gain)
Writech(dJFN,$*n)
WriteS(dJFN,"COMP = ") ; Writech(P>>PutS.Swf.comp)
Writech(dJFN,$*n)
WriteS(dJFN,"DSDATE = ")
for c := 1 to 6 do Writech(dJFN,P>>PutS.Swf.dsdate~c)
Writech(dJFN,$*n)
WriteS(dJFN,"DSTIME = ")
for c := 1 to 8 do Writech(dJFN,P>>PutS.Swf.dstime~c)
Writech(dJFN,$*n)
WriteS(dJFN,"SCALEFACTOR = ")
for c := 1 to 8 do Writech(dJFN,P>>PutS.Swf.scalefactor~c)
Writech(dJFN,$*n)
MODec (dJFN,"STINDEX",P>>PutS.Swf.stindex)
WriteS(dJFN,"DSdate = ")
for c := 1 to 6 do Writech(dJFN,P>>PutS.Swf.DSdate~c")
for c := 1 to 8 do Writech(dJFN,P>>PutS.Swf.DEtime~c)
Writech(dJFN,$*n)
Writech(dJFN,$*n)

!prputs
```

A. SWF-D Program Data

A.1 SWFHEAD, Global Data Definitions

```
// SWF-D Program Header File

get     "<BCPL>HEAD.BCP"
get     "<BCPL>JSHEAD.BCP"
get     "<BCPL>UTILHEAD.BCP"
get     "<BCPL>BDSUBRHEAD.BCP"
get     "<BCPL>STRINGHEAD.BCP"
get     "<BCPL>PSIHEAD.BCP"
get     "<BCPL>OPENFILEHEAD.BCP"

global  {gl

 CopyString:8135
 TaskJFN:2000
 EventJFN:2001
 ScriptJFN:2003
 Logpg:2004
 CurrentFile:2008
 TenexFile:2009
 DCicp:2010
```

```
 Stations:2011
 WorkSchedule:2012
 DaysPerMonth:2013
 InttoTime:2014
 TimetoInt:2015
 R:2016
 P:2017
 ESFPut:2018
 SWFPut:2019

}gl



manifest        {ma

// generally useful constants

 failed          :=      1
 ofOutputNew     :=      ofOutput\ofNewFile

 POINT7x0        :=      #440700 // to avoid expense of BCPL POINT routine
 POINT7x6        :=      #350700 // e.g., instead of  ptr := POINT(7,stgvec,6)
 POINT8x0        :=      #441000         // write  ptr := (POINT7x6,,stgvec)
 POINT8x7        :=      #341000
 POINT36x0       :=      #444400

// time <-> integer conversion codes

 HoursCode       :       3
 MinutesCode     :       2
 SecondsCode     :       1
 CSecsCode       :       0
```

```
// Waveform Component Constants

vertical          :       1
north             :       2
east              :       3
all               :       4

// Task Status Constants

 InitCompleted    :=      2
 InGetEvents      :=      3
 InGetArrivals    :=      5
 AppendingSWF     :=      1
 UpdatingESF      :=      9
 InLimbo          :=      4
 GeneratingSegMap:=       8
 EndGenSegMap     :=      6
 EndGetArrivals   :=      7

// NextTask encodement

 Limbo            :=      1
 GetArrivals      :=      2
 AppendSWF        :=      3
 UpdateESF        :=      4
 GenSegAvailMap   :=      7
 TopoftheQueue    :=      5
 Restart          :=      6


// CheckDC error status codes

 TenexLoad        :=      1
 HardwareProblem:=        2
 DCQuestionable   :=      3
 TBMstatus        :=      4
```

```
 NotEnoughTimeLeft:=    5
 AbnormalDCState:=      6
 NotListening   :=      7

// structure limits

 Tix            :=      100     // limit of task queue
 Esfix          :=      999
 Swfix          :=      999


// MARK program driver constants

 TaskStatus     :=      1
 StationsStatus :=      2
 ESFStatus      :=      3
 UPDATStatus    :=      4

// StationData structure limits

 StationMax     :=      100

}ma

structure { string { n byte; c^511 byte } overlay { stringword^128 word } }

structure       {       TD      {       digit^8         byte }    }
```

```
// Date structure definition

structure       { Date  {
  Yr     bit 18
  Mo     byte
  Day    byte     } }


// Logpg vector structure

structure       { Log {
  Taskix        word
  Tasklim       word
  Task^Tix      word
  LoadLimit     word
  Interval      word
  NextTask      word
  ESFCurrentDate          { Yr     bit 18
                            Mo     byte
                            Day    byte     }
  EventsYear    word
  EventsDay0    word

} }
```

```
// Stations vector structure

structure       { StationData {
  Stationix     word
  AllStationsASLDate    { Yr    bit 18
                          Mo    byte
                          Day   byte    }
  AllStationsSPDETDate  { Yr    bit 18
                          Mo    byte
                          Day   byte    }
  Station^StationMax    { SName^2       word
                          FromDate      { Yr      bit 18
                                          Mo      byte
                                          Day     byte }
                          ToDate        { Yr      bit 18
                                          Mo      byte
                                          Day     byte }
                          SPDETDate     { Yr      bit 18
                                          Mo      byte
                                          Day     byte }
                        }
} }
```

```
structure          { Req {
  eindex           word
  eventnum^9       char
                   fill word
  aindex           word
  sta^5            char
  chantype         char
  rate^2           char
  chanid^4         char
  gain             char
  comp             char
  dsdate^6         char              // datasegstart date
  dstime^8         char
  padate^6         char              // phase arrival date
  patime^8         char
  phaseid^6        char
  amp^7            char
                   fill word
} }
```

```
structure       { EsfL {
  EsfCount      word
  Esf^Esfix     {           eindex          word
                            aindex          word
                            dsdate^6        char
                            dstime^8        char
                                 fill word
                }
} }
```

```
structure       { SwfL {
  SwfCount      word
  Swf^Swfix     {       evdate^5          char
                        evnum^4           char
                          sta^5           char
                          chantype        char
                          rate^2          char
                          chanid^4        char
                          gain            char
                          comp            char
                          dsdate^6        char
                          dstime^8        char
                          scalefactor^8   char
                          staname^5       char
                                fill word
                          starti          word
                          endi            word
                          typ             word   }
} }
```

```
structure       { PutL {
  Esf    { EsfCount        word
           eindex          word
           aindex          word
           dsdate^6        char
           dstime^8        char
                           fill word
}
  Swf    { SwfCount        word
           evdate^5        char
           evnum^4         char
           sta^5           char
           chantype        char
           rate^2          char
           chanid^4        char
           gain            char
           comp            char
                           fill word
           dsdate^6        char
           dstime^8        char
           scalefactor^8   char
           staname^5       char
                           fill word
           starti          word
           endi            word
           typ             word  }
}  }
```

```
structure       { PutS {
  Esf    { EsfCount       word
           eindex         word
           aindex         word
           dsdate^6       char
           dstime^8       char
                  fill word  }
  Swf    { SwfCount       word
           evdate^5       char
           evnum^4        char,
           sta^5          char
           chantype       char
           rate^2         char
           chanid^4       char
           gain           char
           comp           char
                          fill word
           dsdate^6       char
           dstime^8       char
           scalefactor^8  char
                          fill word
           stindex        word
           DSdate^6       char
           DStime^8       char
           DEtime^8       char
           fill word  }
.} }
```

```
// Structure definition for handling SSPDET port records

structure       { SSPDET {
  standex       word
  stime^8       char
  etime^8       char }
}
```

```
// Structure declarations for JFN Mode Word and CCOC words

structure       // NOTE: 'Q xxx' means: true => xxx, false => ~xxx
  { TT                                  // changed by this JSYS: v v
    { OSP  bitb         // Q suppress output                   SFMOD
      MFF  bitb         // Q has mechanical form feed          STPAR
      TAB  bitb         // Q has mechanical tab                STPAR
      LCA  bitb         // Q has lower case                    STPAR
      LEN  bit 7        // page length                         STPAR
      WID  bit 7        // page width                          STPAR
      WAK  bit 6        // wakeup control                      SFMOD
        overlay                 // Q wakeup on:
          {       bit 2         // (not used)
            WKF  bitb           // formatting control character
            WKN  bitb           // non-formatting control char
            WKP  bitb           // punctuation character
            WKA  bitb           // alphanumeric character
          }
      ECH  bit 2        // (echoing) Tenex: values 0-3         SFMOD
        overlay         // Tops-20 only:
          { ECO  bitb           // Q echo on                   SFMOD
            ECM  bitb           // (F|T) => (deferred|immediate) STPAR
          }
      ALK  bitb         // Q accept links                      TLINK
      AAD  bitb         // Q accept advice                     TLINK
      DAM  bit 2        // terminal data modes (see codes)     SFMOD
      UOC  bitb         // UC output control (Q indicate as 'X)  STPAR
                  // Tenex: LC output ctrl (Q indicate as %X) STPAR
      LIC  bitb         // LC input control (Q cnvt LC to UC)    STPAR
      DUM  bit 2        // duplex mode (see codes)             STPAR
      PGM  bitb         // Q page mode output              STPAR
                  // Tenex: Q Repeat Last Character          (BKJFN)
      CAR  bitb         // carrier state (Q dataset & carrier on) --
  } }
```

A.2 SWFAlo, Storage and Work Areas

```
//  SWFAlo - storage & working areas

get     "<CCA-SWF>SWFHEAD.BCP"

static  {stat                       // best aligned on page boundary

  ESFPut          :       vec     6000    // max length really 5995
  SWFPut          :       vec     16000   // max length really 15985
  CurrentFile     :       vec     100     // a BCPL string of the form
                                          //  Ynnnn.Mnn.Dnn
  TenexFile       :       vec     100     // a BCPL string of the form
                                          // Ynnnn%Mnn%Dnn
  TaskJFN         :       0
  EventJFN        :       0
  ScriptJFN       :       0
  DCicp           :       nil             // true|false => connected to DC
  Stations        :       vec     512     // mapped into via the StationData
                                          // structure
  Logpg           :       vec     512     // mapped into via the Log structure
  DaysPerMonth    :       table nil,31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
//                                 Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

  R               :       vec     512     // request area for arrivals
  P               :       vec     512     // put area - mapped from R
}stat
```

References

"CCA-78-10 Program Design for SWF-D: The Signal Waveform File Demon", Donald E. Eastlake, III and Joanne Z. Sattley, 30 June 1978, Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts 02139.

"Datacomputer Technical Bulletin Number 2, DCSUBR: Functional Specifications", Jerry Farrell, July 1976, Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts 02139.

"Datacomputer Technical Bulletin Number 8, The CCA Datacomputer Status Server", Donald E. Eastlake, III, April 1978, Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts 02139.

"Datacomputer Version 5 User Manual", July 1978, Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts 02139.

"CCA-79-09 Report on the Implementation and Test of SWF-D: The Signal Waveform File Demon", Joanne Z. Sattley, 31 January 1979, Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts 02139.

"BCPL Manual", September 1974, Bolt, Beranek and Newman, Inc., 50 Moulton Street, Cambridge, Mass. 02138.

"VELA NETWORK Mass Store Data Retrieval Guide", Emily B. McCoy and Edwin W. Meyer, Jr., 31 January 1977, Seismic Data Analysis Center, Teledyne Geotech, Alexandria Laboratories, Alexandria, Virginia 22313.

"VELANET ESF/SWF PROCESSING", Joseph Greenhalgh, Design Draft, 1 February 1978, Teledyne Geotech, Alexandria Laboratories, P. O. Box 334, Alexandria, Virginia 22313.

"ESD-TR-78-64 Semiannual Technical Summary, Seismic Discrimination", 31 March 1978, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, Massachusetts 02173.